Quantum Algorithm for Path-Edge Sampling

Stacey Jeffery
QuSoft and CWI, Amsterdam, The Netherlands

Shelby Kimmel
Middlebury College, VT, USA

Alvaro Piedrafita
QuSoft and CWI, Amsterdam, The Netherlands

Abstract

We present a quantum algorithm for sampling an edge on a path between two nodes s and t in an undirected graph given as an adjacency matrix, and show that this can be done in query complexity that is asymptotically the same, up to log factors, as the query complexity of detecting a path between s and t. We use this path sampling algorithm as a subroutine for st-path finding and st-cut-set finding algorithms in some specific cases. Our main technical contribution is an algorithm for generating a quantum state that is proportional to the positive witness vector of a span program.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Quantum query complexity; Theory of computation → Algorithm design techniques

Keywords and phrases Algorithm design and analysis, Query complexity, Graph algorithms, Span program algorithm, Path finding, Path detection

Digital Object Identifier 10.4230/LIPIcs.TQC.2023.5

Related Version *Full Version*: `https://arxiv.org/pdf/2303.03319.pdf`

Funding *Shelby Kimmel and Stacey Jeffery:* sponsored by the U.S. Army Research Office and this work was accomplished under Grant Number W911NF-20-1-0327. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.
Stacey Jeffery: supported by NWO Klein project number OCENW.Klein.061, and the European Union (ERC, ASC-Q, 101040624). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. SJ is a CIFAR Fellow in the Quantum Information Science Program.

Acknowledgements We thank Jana Sotáková and Mehrdad Tahmasbi for insightful discussions about path finding via edge sampling.

1 Introduction

Finding and detecting paths between two vertices in a graph are important related problems, both in and of themselves, and as subroutines in other applications, but there is still much to understand in this area. While classically these problems seem to be equivalent, an intriguing question is whether the same holds for quantum algorithms: there are cases where a quantum algorithm can *detect* a path between s and t in significantly less time than any known quantum algorithm takes to *find* such a path. In particular, path finding on a glued trees graph is one of Aaronson's top ten open problems in query complexity [12, 1], as the best known quantum algorithms that find an st-path in such graphs have exponentially worse running time than the best quantum algorithms for detecting one, and

understanding how these problems are related could improve our understanding of why quantum computers achieve dramatic speedups for certain problems. As an example of more immediate practical interest: path finding in supersingular isogeny graphs is one approach to attacking cryptosystems based on supersingular isogenies [11, 16], but currently the best known attack of this form still takes exponential time [36] (see also [19]).

In this paper, we consider the quantum query complexity of a somewhat intermediate problem: finding an edge on an st-path in an undirected graph.[1] In the classical case, it seems hard to imagine how one could find an edge on an st-path without first finding an st-path, but we show that in the quantum case, one can sample an st-path edge with similar resources to what is needed to detect the existence of an st-path. In some cases, this can be done with significantly fewer queries than the best previously known path-finding algorithms. We show this ability to sample an edge on a path has some useful applications, including to sabotaging networks (finding st-cut sets) and to finding paths in certain graphs faster than existing path finding algorithms.

Previously, Dürr, Heiligman, Høyer and Mhalla [18] described an algorithm for connectivity in the adjacency matrix model that uses $O(n^{3/2})$ queries for an n-vertex graph. Their algorithm works by keeping track of known connected components, and then uses a quantum search to look for any edge that connects any two components previously not known to be connected. While the authors use this algorithm to decide connectivity, we note that after $O(n^{3/2})$ queries, the algorithm will produce (with high probability) a list of the connected components of the graph, as well as a set of edges for each component that is a witness to that component's connectivity (a spanning tree). This data can then be used to find a path from s to t, if s and t are in the same component. This algorithm uses $O(\log n)$ qubits and $O(n \log n)$ classical bits, and applies to both directed and undirected graphs.

However, the algorithm of Dürr et al. does not take advantage of any structure in the graph. This is in contrast to an undirected path *detection* quantum algorithm of Belovs and Reichardt [7], further analyzed and refined in [22, 2], which, for example, can detect a path between vertices s and t with $\widetilde{O}(\sqrt{L}n)$ adjacency matrix queries when there is an st-path of length L, and even better in the case of multiple short paths, or in the case of certain promises when there is no path. In fact, there are even sufficiently structured promises on the input for which this algorithm performs superpolynomially better than the best possible classical algorithm [25]. While this path detection algorithm runs faster than $O(n^{3/2})$ in many cases, the algorithm does not output any information about the st-path – it simply determines whether a path exists.

Our contribution is an algorithm that reproduces the query complexity of the Belovs-Reichardt undirected path detection algorithm, even for structured inputs – for example, our algorithm uses $\widetilde{O}(\sqrt{L}n)$ queries when there is a path of length L – but now returns *some* information about edges on an st-path: namely, a path edge.[2] Specifically, our algorithm outputs an st-path edge sampled with probability that depends on the optimal st-flow between s and t. This is how electrons would flow in an electrical network if edges in the graph were replaced by wires with resistors and a battery were connected between s and t. For intuition, an edge is more likely to be sampled if it is on *more* or *shorter* paths. Thus,

[1] In this paper, we use *path* to refer to a self-avoiding path, meaning a path with no repeated vertices.

[2] As we hinted at with our statement of advantages for the Belovs-Reichardt algorithm in the case of shorter and/or multiple paths, the Belovs-Reichardt algorithm for st-path detection actually has a complexity that depends on the structure of the graph in a more subtle way, replacing L with an upper bound on the *effective resistance* between s and t, which is *at most* the length of the shortest path between s and t. This more subtle analysis also applies to our edge finding algorithm.

in the case of a single path between s and t, our algorithm samples each edge in the path with equal probability (up to some error in total variation distance). When there are disjoint paths of different lengths, our algorithm is more likely to sample an edge on a short path than a long path – the probability of sampling from a particular path of length ℓ is proportional to $1/\ell$. (This means, unfortunately, that if there are many long paths, we might still be more likely to sample an edge on some long path than an edge on a short path). We prove that finding an st-path edge classically requires $\Omega(n^2)$ queries in the worst case, even if promised that there is a path of length L, as long as $L \geq 3$.

With the ability to quickly find edges on short paths, we can create an improved algorithm for *finding* st-paths in undirected graphs with a unique, short st-path. Given an adjacency matrix for an n-vertex graph, if there is a unique st-path, whose (possibly unknown) length is L, we can find all of the edges in the path in $\widetilde{O}(L^{1+o(1)}n)$ expected queries. When $L = o(\sqrt{n})$, this is an improvement over the Dürr et al. algorithm. In the general case that there is more than one st-path, we prove that we can find all edges in a single path in $\widetilde{O}(L^{3/2}n)$ queries when L is the (possibly unknown) length of the *longest* path (although our approach in this case does not use the edge sampling algorithm as a subroutine). When $L = o(n^{1/3})$, this is an improvement over the Dürr et al. algorithm.

We additionally use our sampling algorithm to find st-cut sets, in the case that s and t are each part of a highly connected component, and there are only a few edges connecting those components. Because these few connecting edges are bottlenecks in the flow, there will be a lot of flow over those connecting edges, and so a high probability of sampling them, and hence finding an st-cut set. We describe a particular family of n-vertex graphs were we can find such a cut set in $\widetilde{O}(n)$ queries, where any classical algorithm would require $\Omega(n^2)$ queries.

Our edge sampling algorithm is a special case of a new span-program-based algorithm (Section 3) for generating quantum states called *span program witness states* (or simply *witness states*). One of the key elements of the analysis of span program algorithms for deciding Boolean functions [34] is the positive witness (see Definition 2), which is a vector that witnesses that the function evaluates a particular input to 1. While in the usual span program algorithm, the output on input x is $f(x)$, in our case, we output a quantum state proportional to the positive witness for input x. In the case of the Belovs-Reichardt span program for st-connectivity [7], a positive witness is a linear combination of edges that are on paths between s and t, where the amplitudes depend on the optimal st-flow (see Definition 4). Generating and then measuring such a state allows us to sample st-path edges.

Our results more generally hold for the case where the input x defines a subgraph $G(x)$ of some arbitrary graph G, that is not necessarily a complete graph. Although we do not attempt to analyze time complexity in this work, we suspect that our query algorithms on graphs are also time efficient when there is an efficient way to perform a quantum walk on the underlying graph G, as in [25]. For example, when G is the n-vertex complete graph (i.e. the oracle allows you to query elements of the full adjacency matrix for a n-vertex graph, as we have been assuming throughout this introduction), there is an efficient way to do this walk, and so in this case the time complexity of our algorithms is likely the same as the query complexity, up to log factors.

1.1 Future Directions

A natural future direction is to try to use our edge finding technique for path finding in more general settings than the ones we consider. One surprising aspect of our algorithm is that it does not necessarily find edges in the order in which they appear in the path, and instead

often finds edges in the middle of a path with high probability. The form of our algorithm thus seems to circumvent a recent lower bound on path-finding in glued trees graphs that applies to algorithms that always maintain a path from the starting node to any vertex in the algorithm's state [13]. However, one reason to be pessimistic for this particular application is that in the glued trees graph, *all* edges connected to the starting vertex are in some st-path. Still, we are hopeful that for some graphs, finding an edge in the middle of some st-path opens up the possibility of new divide-and-conquer approaches for path finding.

We are only able to take advantage of the fact that we sample edges according to the optimal st-flow for very specific graphs, like those with a single path, or with bottleneck flows, but we hope that this edge sampling distribution will prove useful in additional applications. In recent independent work, Apers and Piddock [3] develop a similar edge sampling algorithm in the adjacency list model, which they use to analyze connections between electric flows and quantum walks, and they prove that walks that proceed via their edge sampling algorithm need only logarithmically many rounds before they have a high probability of reaching a target vertex, on trees. We believe that such edge sampling methods will likely find further applications.

We have only applied our span program witness state generation algorithm to the span program for path detection. Span program algorithms exist for a wide range of graph problems, from bipartiteness [8] and cycle detection [8, 17], to triangle [9] and other subgraph detection [30], to other combinatorial search problems [6, 4]. Perhaps the span program witness states for these problems would be useful for certain applications. Beyond span program algorithms, dual adversary algorithms (which are equivalent to span programs for decision problems, but generalize to state conversion problems [31]) and multidimensional quantum walks [27, 23] all have a similar notion of witnesses in their design and analysis. Similar techniques might yield witness generation algorithms for these more general algorithm design paradigms.

We suspect our path finding algorithms are not optimal, as for graphs with longest paths of length $\Omega(n^{1/3})$, our algorithms do not outperform Dürr et al.'s algorithm. We wonder whether it is possible to find paths using $o(n^{3/2})$ queries whenever the longest path has length $o(n)$, or to prove that this is not possible, perhaps by expanding on techniques for lower bounding path-finding on welded trees [13].

Finally, all of our algorithms apply only to undirected graphs, while the algorithm of [18] applies equally well to directed or undirected graphs. While there are span program algorithms for problems on directed graphs (see e.g. [4]), they do not exhibit the same speedups with short or many paths that the undirected span program algorithms possess. It would be interesting to better understand whether there are ways to obtain similar improvements in query complexity for directed graphs.

Organization

In Section 3 we present our main technical result: an algorithm for generating a state proportional to a span program witness for x. In Section 4, we show how to apply this to finding a path edge (Section 4.1), and give an example of a particular family of graphs in which the classical complexity of finding a path edge is quadratically worse than our quantum algorithm (Theorem 20). In Section 4.2, we show how our edge finding algorithm can be applied to efficiently find an st-cut set in a particular family of graphs, and in Section 4.3 we show how it can be applied to find an st-path in $\widetilde{O}(nL^{1+o(1)})$ queries when there is a *unique* st-path of length L (Theorem 25); and also give an algorithm for finding an st-path in general graphs in $\widetilde{O}(nL^{3/2})$ queries when L is the length of the *longest* st-path (Theorem 26).

2 Preliminaries

We first introduce some basic notation. We let $\|\cdot\|$ denote the l_2 norm, $[m] \coloneqq \{1, 2, 3, \ldots, m\}$, and let $\mathcal{L}(H, V)$ denote the set of linear operators from the vector space H to the vector space V.

2.1 Span Programs

Span programs are a linear algebraic model of computation, introduced in [28], that have proven extremely useful for analyzing query [34, 35], space [24], and time complexity [7, 15, 5] in quantum algorithms. We follow Ref. [21] closely in our definitions.

▶ **Definition 1** (Span Program). *For a finite set R, a span program on R^m is a tuple $\mathcal{P} = (H, \mathcal{V}, |\tau\rangle, A)$ where*

1. *H is a direct sum of finite-dimensional inner product spaces: $H = H_1 \oplus H_2 \cdots H_m \oplus H_{true} \oplus H_{false}$, and for $j \in [m]$ and $a \in R$, we have $H_{j,a} \subseteq H_j$, such that $\sum_{a \in R} H_{j,a} = H_j$;*
2. *$\mathcal{V}$ is a vector space;*
3. *$|\tau\rangle \in \mathcal{V}$ is a target vector; and*
4. *$A \in \mathcal{L}(H, \mathcal{V})$.*

Given a string $x \in R^m$, we use $H(x)$ to denote the subspace $H_{1,x_1} \oplus \cdots \oplus H_{m,x_m} \oplus H_{true}$, and we denote by $\Pi_{H(x)}$ the orthogonal projector onto the space $H(x)$.

An important concept in the analysis of span programs and quantum query complexity is that of *witnesses*:

▶ **Definition 2** (Positive Witness). *Given a span program $\mathcal{P} = (H, \mathcal{V}, |\tau\rangle, A)$ on R^m and $x \in R^m$, $|w\rangle \in H(x)$ is a* positive witness *for x in $\mathcal{P}$ if $A|w\rangle = |\tau\rangle$. If a positive witness exists for x, we define the* witness size *of x in $\mathcal{P}$ as*

$$w_+(x) = w_+(\mathcal{P}, x) \coloneqq \min\left\{ \||w\rangle\|^2 : |w\rangle \in H(x) \text{ and } A|w\rangle = |\tau\rangle \right\}. \tag{1}$$

We say that $|w\rangle \in H(x)$ is the optimal positive witness for x if $\||w\rangle\|^2 = w_+(\mathcal{P}, x)$ and $A|w\rangle = |\tau\rangle$.

Our main algorithm produces a normalized version of this unique optimal positive witness, $|w\rangle/\||w\rangle\|$. (To see that the optimal positive witness is unique, for contradiction assume that the optimal positive witness is not unique – then a linear combination of two optimal positive witnesses produces a witness with smaller witness size than either.)

A span program $\mathcal{P}$ encodes a function $f : X \to \{0, 1\}$ in the following way. We say $f(x) = 1$ if x has a positive witness, and $f(x) = 0$ if x does not have a positive witness. We say such a $\mathcal{P}$ decides the function f.

We will also need the concept of an approximate negative witness.

▶ **Definition 3** (Negative Error, Approximate Negative Witness). *Given a span program $\mathcal{P} = (H, \mathcal{V}, |\tau\rangle, A)$ on R^m and $x \in R^m$, we define the negative error of x in $\mathcal{P}$ as*

$$e_-(x, \mathcal{P}) \coloneqq \min\left\{ \|\langle\widetilde{\omega}|A\Pi_{H(x)}\|^2 : \langle\widetilde{\omega}| \in \mathcal{L}(\mathcal{V}, \mathbb{R}), \langle\widetilde{\omega}|\tau\rangle = 1 \right\}. \tag{2}$$

Note that $e_-(x, \mathcal{P}) = 0$ if and only if $\mathcal{P}$ decides a function f with $f(x) = 0$. Any $\langle\widetilde{\omega}|$ such that $\|\langle\widetilde{\omega}|A\Pi_{H(x)}\|^2 = e_-(x, \mathcal{P})$ is called an approximate negative witness for x in P. We define the approximate negative witness size of x as:

$$\widetilde{w}_-(x, \mathcal{P}) \coloneqq \min\left\{ \|\langle\widetilde{\omega}|A\|^2 : \langle\widetilde{\omega}| \in \mathcal{L}(\mathcal{V}, \mathbb{R}), \langle\widetilde{\omega}|\tau\rangle = 1, \|\langle\widetilde{\omega}|A\Pi_{H(x)}\|^2 = e_-(x, \mathcal{P}) \right\}. \tag{3}$$

We call an approximate negative witness $\langle\widetilde{\omega}|$ that also minimizes $\|\langle\widetilde{\omega}|A\|^2$ an optimal approximate negative witness.

We use the following notation for maximum positive and approximate negative witness sizes:

$$W_+(\mathcal{P}, f) = W_+ \coloneqq \max_{x \in f^{-1}(1)} w_+(\mathcal{P}, x), \qquad \widetilde{W}_-(\mathcal{P}, f) = \widetilde{W}_- \coloneqq \max_{x \in f^{-1}(1)} \widetilde{w}_-(x, \mathcal{P}). \tag{4}$$

Note that we are restricting to 1-inputs of f. That is because our witness generation algorithm will assume that x is a 1-input, unlike previous span-program-based algorithms that *decide* f.

2.2 Quantum Query Algorithms

The algorithms we develop are query algorithms, where we can access a unitary oracle O_x for some $x \in X \subseteq R^m$ such that O_x acts on the space $\mathbb{C}^m \otimes \mathbb{C}^q$ as $O_x|i\rangle|a\rangle = |i\rangle|x_i + a \mod q\rangle$. where $q = |R|$, x_i is the value of the i^{th} element of x and $|i\rangle \in \mathbb{C}^m$ and $|a\rangle \in \mathbb{C}^q$ are standard basis states.

The query complexity of an algorithm is the number of times O_x must be used, in the worst case over $x \in X$. In our case, we will also consider the expected query complexity on input x, which is the average number of times O_x must be used when given a particular input x, where the randomness is due to random events in the course of the algorithm.

2.3 Graph Theory and Connection to Span Programs

Let $G = (V, E)$ be an undirected graph.[3] We will particularly consider graphs with specially labeled vertices $s, t \in V$, such that there is a path from s to t in G. Let $\overrightarrow{E} = \{(u, v) : \{u, v\} \in E\}$; that is $\overrightarrow{E}$ is the set of directed edges corresponding to the edges of G. Given a graph $G = (V, E)$, for $u \in V$, we denote by G_u^- the subgraph of G on the vertices $V \setminus \{u\}$, and with overloading of notation for $S \subseteq E$, we denote by G_S^- the subgraph of G with edges S removed. (It will be clear from context whether we are removing edges or vertices from the graph.)

On a graph G with s and t connected we will consider a *unit st-flow*, which is a linear combination of cycles and st-paths, formally defined as a function on $\overrightarrow{E}$ with the following properties.

▶ **Definition 4** (Unit *st*-flow). *Let $G = (V, E)$ be an undirected graph with $s, t \in V(G)$, and s and t connected. Then a* unit *st*-flow *on G is a function $\theta : \overrightarrow{E} \to \mathbb{R}$ such that:*

1. *For all $(u, v) \in \overrightarrow{E}$, $\theta(u, v) = -\theta(v, u)$;*
2. *$\sum_{v:(s,v)\in\overrightarrow{E}} \theta(s, v) = \sum_{v:(v,t)\in\overrightarrow{E}} \theta(v, t) = 1$; and*
3. *for all $u \in V \setminus \{s, t\}$, $\sum_{v:(u,v)\in\overrightarrow{E}} \theta(u, v) = 0$.*

▶ **Definition 5** (Unit Flow Energy). *Given a graph $G = (V, E)$ and a unit st-flow θ on G, the* unit flow energy *of θ is $J(\theta) = \frac{1}{2}\sum_{e\in\overrightarrow{E}} \theta(e)^2$.*

▶ **Definition 6** (Effective resistance). *Let $G = (V, E)$ be a graph with $s, t \in V$. If s and t are connected in G, the* effective resistance *of G between s and t is $R_{s,t}(G) = \min_\theta J(\theta)$, where θ runs over all unit st-unit flows of G. If s and t are not connected in G, $R_{s,t}(G) = \infty$.*

[3] Our results easily extend to multigraphs, see [22], but for simplicity, we will not consider multigraphs here.

Interpretation of the optimal flow

The st-flow with minimum energy is unique, and describes the electric current going through that edge if the graph represents a network of unit resistors and we put a potential difference between s and t. The minimum energy flow has several other interpretations and connections to other graph properties. For reference, and for those who would like to build their intuition for this object, we have collected some of these relationships in the full version of this work [26, Appendix A].

Graph access

We turn graph problems into oracle problems by letting a string $x \in \{0,1\}^m$ specify a subgraph $G(x)$ of G. In particular, we associate each edge $e \in E$ with a number in $[m]$. Then, given a string $x \in \{0,1\}^m$, let $G(x) = (V, E(x))$ be the subgraph of G that contains an edge $e \in E$ if e is associated with the integer $i \in [m]$ and $x_i = 1$, where x_i is the ith bit of x. In this oracle problem, one is given access to an oracle O_x for x (or classically, given the ability to query the values of the bits of x one at a time), and a description of the parent graph G along with the association between bits of x and edges of G, and the goal is to determine something about the graph $G(x)$ using as few queries as possible. Let $E_i \subset E$ be the set of edges associated with the ith bit of x. When not specified otherwise, one should assume that $m = |E|$, and then associate each edge of G uniquely with a bit of the input string. In this case, when G is the complete graph, O_x is equivalent to query access to the adjacency matrix of a graph. When we consider subgraphs of the original graph (like G_u^-), we assume that the edges are associated with the same indices as in the original graph, unless otherwise specified.

Most of the applications in this paper are related to the problem of detecting a path between s and t – more commonly called st-connectivity. We define $st\text{-CONN}_G(x) := 1$ if s and t are connected in $G(x)$, and 0 otherwise. The following span program, which we denote by $\mathcal{P}_{G_{st}}$, first introduced in Ref. [28] and used in the quantum setting in Ref. [7], decides $st\text{-CONN}_G(x)$: for a graph $G = (V, E)$, where $m = |E|$, define the span program $\mathcal{P}_{G_{st}}$ as:

$$\begin{aligned}
&\forall i \in [m], H_{i,1} = \text{span}\{|(u,v)\rangle : \{u,v\} \in E_i\}, H_{i,0} = \emptyset \\
&\mathcal{V} = \text{span}\{|v\rangle : v \in V(G)\} \\
&|\tau\rangle = |s\rangle - |t\rangle \\
&\forall (u,v) \in \overrightarrow{E} : \ A|u,v\rangle = |u\rangle - |v\rangle.
\end{aligned} \tag{5}$$

For $\mathcal{P}_{G_{st}}$, the negative approximate witness size is bounded by $\widetilde{W}_- = O(n^2)$ [21]. If s and t are connected in $G(x)$, the optimal positive witness of x in $\mathcal{P}_{G_{st}}$ is [7, 22]

$$|\theta^*\rangle = \frac{1}{2} \sum_{e \in \overrightarrow{E}} \theta^*(e)|e\rangle, \tag{6}$$

where θ^* is the st-unit flow with minimal energy, so by Definitions 2 and 6, $w_+(\mathcal{P}_{G_{st}}, x) = \frac{1}{2} R_{s,t}(G(x))$.

One of our main applications is to apply our witness state generation algorithm to the span program $\mathcal{P}_{G_{st}}$, in which case, we produce a quantum state close to $|\theta^*\rangle/\||\theta^*\rangle\|$ where θ^* is the optimal unit st-flow on $G(x)$. If we were to create $|\theta^*\rangle/\||\theta^*\rangle\|$ exactly, and then measure in the standard basis, the probability that we obtain the edge e is $\theta^*(e)^2/(2R_{s,t}(G(x)))$. Let $q_{G(x),s,t}$ denote the distribution such that for $\forall e \in \overrightarrow{E}$,

$$q_{G(x),s,t}(e) = \theta^*(e)^2/(2R_{s,t}(G(x))). \tag{7}$$

Additionally, this optimal flow θ^* is a convex combination of (self-avoiding) st-paths, as we prove in the full version of this work [26, Appendix A]:

▶ **Lemma 7.** *An* st-path *in* $G(x)$ *is a sequence of* distinct *vertices* $\vec{u} = (u_0, \ldots, u_\ell)$ *such that* $s = u_0$, $t = u_\ell$, *and for all* $i \in [\ell]$, $(u_{i-1}, u_i) \in \overrightarrow{E}(G(x))$. *From* $\vec{u}$, *we define*

$$|\rho_{\vec{u}}\rangle = \frac{1}{\sqrt{2}} \sum_{i=0}^{\ell-1} (|u_i, u_{i+1}\rangle - |u_{i+1}, u_i\rangle) \tag{8}$$

and refer to all such states as st-path states of $G(x)$. *Then if* $|\theta^*\rangle$ *is the optimal positive witness for* x *in* $\mathcal{P}_{G_{s,t}}$, *it is a linear combination of* st*-path states in* $G(x)$.

A final pair of tools we use are a quantum algorithm that decides st-$\text{CONN}_G(x)$ with fewer queries in the case of small effective resistance, without knowing the effective resistance ahead of time, and a quantum algorithm for estimating the effective resistance:

▶ **Lemma 8** ([2])**.** *Fix* $\delta > 0$ *and a family of* n*-vertex graphs* G *with vertices* s *and* t. *Then there is a quantum algorithm* $\texttt{PathDetection}(O_x, G, s, t, \delta)$ *such that,*

1. *The algorithm returns* st-$\text{CONN}_G(x)$ *with probability* $1 - O(\delta)$.
2. *On input* x, *the algorithm uses* $O\left(n\sqrt{R_{s,t}(G(x))} \log\left(\frac{n}{R_{s,t}(G(x))\delta}\right)\right)$ *expected queries if* st-$\text{CONN}_G(x) = 1$, *and* $O\left(n^{3/2} \log 1/\delta\right)$ *expected queries if* st-$\text{CONN}_G(x) = 0$.

▶ **Lemma 9** ([21])**.** *Fix* $\delta > 0$ *and a family of* n*-vertex graphs* G *with vertices* s *and* t. *Then there is a quantum algorithm* $\texttt{WitnessSizeEst}(O_x, G, s, t, \epsilon, \delta)$ *that, on input* x *such that* st-$\text{CONN}_G(x) = 1$, *with probability* $1 - \delta$, *outputs an estimate* $\hat{R}$ *for* $R_{s,t}(G(x))$ *such that*

$$\left|\hat{R} - R_{s,t}(G(x))\right| \leq \epsilon R_{s,t}(G(x)), \tag{9}$$

using $\widetilde{O}\left(\sqrt{\frac{R_{s,t}(G(x))n^2}{\epsilon^3}} \log(1/\delta)\right)$ *expected queries; and on input* x *such that* st-$\text{CONN}_G(x) = 0$, *uses at most* $\widetilde{O}\left((n/\epsilon)^{3/2} \log(1/\delta)\right)$.

Lemma 9 is a special case of [21, Theorem 3.8], which gives an algorithm for estimating the quantity $w_+(x)$ from *any* span program. If we apply this construction with the span program $\mathcal{P}_{G_{s,t}}$, we can estimate its positive witness sizes, which are precisely $\frac{1}{2}R_{s,t}(G(x))$. The algorithm described in [21, Theorem 3.8] assumes that the input is a 1-input to st-$\text{CONN}_G(x)$, but can easily be modified to always stop after at most $\widetilde{O}\left((n/\epsilon)^{3/2} \log(1/\delta)\right)$ steps, regardless of the input, since $R_{s,t}(G(x)) \leq n$. The algorithm as stated also only works with bounded error, but the success probability can be amplified to $1 - \delta$ by repeating $\log(1/\delta)$ times and taking the median estimate.

3 Witness Generation

Our main technical result, on generating span program witness states is the following:

▶ **Theorem 10.** *Given a span program* $\mathcal{P}$ *that decides a function* f, *and constants* ϵ, δ, *there is an algorithm (Algorithm 1) that, given as input an oracle* O_x *such that* $f(x) = 1$ *with optimal positive witness* $|w\rangle$, *outputs a state* $|\hat{w}\rangle/\||\hat{w}\rangle\|$ *such that* $\left\||w\rangle/\sqrt{w_+(x)} - |\hat{w}\rangle/\||\hat{w}\rangle\|\right\|^2 \leq O(\epsilon)$ *with probability* $1 - O(\delta)$, *and uses* $\widetilde{O}\left(\sqrt{\frac{w_+(x)\widetilde{W}_-}{\epsilon}} \log\left(\frac{1}{\delta}\right)\right)$ *expected queries to* O_x.

For comparison, a span program algorithm can *decide* f with bounded error in expected query complexity $\widetilde{O}\left(\sqrt{w_+(x)\widetilde{W}_-}\right)$, so Theorem 10 gives a matching complexity for generating a witness state. As we will see in Section 4.1, in the case of the span program $\mathcal{P}_{G_{s,t}}$ for st-connectivity on subgraphs of G, this implies that we can sample an st-path edge in the same complexity used by the span program algorithm to decide if an st-path exists.

A key subroutine for our witness state generation algorithm will be quantum phase estimation. In quantum phase estimation one implements a controlled version of a unitary U acting on a Hilbert space $\mathcal{H}_A$ on an input state $|\psi\rangle \in \mathcal{H}_A$. The state $|\psi\rangle$ can be decomposed into its eigenbasis with respect to U as $|\psi\rangle = \sum_i \alpha_i|\lambda_i\rangle$, where $U|\lambda_i\rangle = e^{i\phi_i\pi}$ and we say ϕ_i is the phase of the state $|\lambda_i\rangle$. Then when phase estimation is performed with precision Θ the probability that you measure a phase of 0 after the phase estimation procedure is approximately given by $\sum_{i:|\phi_i|\leq\Theta}|\alpha_i|^2$, and the non-normalized state that results after measuring a phase of 0 is approximately $\sum_{i:|\phi_i|\leq\Theta}\alpha_i|\lambda_i\rangle$. In other words, phase estimation can be used to project into the low phase space (with phase less than Θ) with probability that depends on the amount of amplitude the original state had on low-phase eigenstates. For an accuracy parameter ϵ, the number of uses of U in phase estimation scales as $O\left(\frac{1}{\Theta}\log\frac{1}{\epsilon}\right)$. A more rigorous description of the guarantees of phase estimation is given below in Lemma 11.

The basic idea of the algorithm that we use to prove Theorem 10 is to apply phase estimation with a unitary $U(\mathcal{P},x,\alpha)$, (which can be implemented with access to an oracle O_x and depends on a span program $\mathcal{P}$, and a positive real parameter α), on a state $|\hat{0}\rangle$. We show that the eigenspectrum of $|\hat{0}\rangle$ relative to $U(\mathcal{P},x,\alpha)$ decomposes into two states, $|\hat{0}\rangle \oplus \frac{1}{\alpha}|w\rangle$, which is a 0-phase eigenstate of $U(\mathcal{P},x,\alpha)$, and $|\psi_{x,+}\rangle$, which has small overlap with the low-phase space of $U(\mathcal{P},x,\alpha)$.

If we do phase estimation with $U(\mathcal{P},x,\alpha)$ on $|\hat{0}\rangle$ with sufficiently small precision, and then if we measure a phase of 0, as discussed above, we will approximately project into the state $|\hat{0}\rangle \oplus \frac{1}{\alpha}|w\rangle$. From there, if we make the measurement $\{|\hat{0}\rangle\langle\hat{0}|, I-|\hat{0}\rangle\langle\hat{0}|\}$, and obtain outcome $I-|\hat{0}\rangle\langle\hat{0}|$ the state will project into $|w\rangle$, as desired.

Next, there comes a balancing act for our choice of α. When α is too small, $|\hat{0}\rangle$ has small overlap with the span of $|\hat{0}\rangle \oplus \frac{1}{\alpha}|w\rangle$, so we are not very likely to measure a phase of 0 when we do phase estimation with $U(\mathcal{P},x,\alpha)$ on $|\hat{0}\rangle$. However, when α gets too large, while it becomes very likely to measure a phase of 0 and thus obtain the state $|\hat{0}\rangle \oplus \frac{1}{\alpha}|w\rangle$, we will be unlikely to subsequently measure outcome $I-|\hat{0}\rangle\langle\hat{0}|$.

The sweet spot is when $\alpha \approx \sqrt{w_+(x)}$, in which case both measurement outcomes we require have a reasonable probability of occurring. Since we don't know $w_+(x)$ ahead of time, we must first estimate an appropriate value of α to use, which we do by iteratively testing larger and larger values of α.[4] Our test involves estimating the probability of measuring a phase of 0 when phase estimation with $U(\mathcal{P},x,\alpha)$ is performed on $|\hat{0}\rangle$, which we show provides an estimate of $\alpha/\sqrt{w_+(x)}$.

3.1 Proof of Theorem 10

Before introducing the algorithm we use to prove Theorem 10, we introduce some key concepts, lemmas, and theorems that will be used in the analysis.

Let $\tilde{H} = H \oplus \text{span}\{|\hat{0}\rangle\}$, and $\tilde{H}(x) = H(x) \oplus \text{span}\{|\hat{0}\rangle\}$, where $|\hat{0}\rangle$ is orthogonal to H. Then we define $\tilde{A}^\alpha \in \mathcal{L}(\tilde{H},\mathcal{V})$ as

$$\tilde{A}^\alpha = \frac{1}{\alpha}|\tau\rangle\langle\hat{0}| - A. \tag{10}$$

[4] There is a similar algorithm in [21] that estimates $w_+(x)$, but it is more precise than we require.

Let $\Lambda^\alpha \in \mathcal{L}(\tilde{H}, \tilde{H})$ be the orthogonal projection onto the kernel of $\tilde{A}^\alpha$, and let $\Pi_x \in \mathcal{L}(\tilde{H}, \tilde{H})$ be the orthogonal projector onto $\tilde{H}(x)$. Finally, let $U(\mathcal{P}, x, \alpha) = (2\Pi_x - I)(2\Lambda^\alpha - I)$. Note that $2\Pi_x - I$ can be implemented with two applications of O_x [21, Lemma 3.1], and $2\Lambda^\alpha - I$ can be implemented without any applications of O_x.

We will use parallelized phase estimation, as described in Ref. [32], which provides improved error bounds over standard phase estimation. In particular, given a unitary U acting on a Hilbert Space $\mathcal{H}$, a precision $\Theta > 0$, and an accuracy $\epsilon > 0$, we can create a circuit $D(U)$ that implements $O(\log \frac{1}{\epsilon})$ parallel copies of the phase estimation circuit on U, each to precision $O(\Theta)$, that each estimate the phase of a single copy of a state $|\psi\rangle$. That is, $D(U)$ acts on the space $\mathcal{H}_A \otimes ((\mathbb{C}^2)^{\otimes b})_B$ where $b = O\left(\log \frac{1}{\Theta} \log \frac{1}{\epsilon}\right)$, and A labels the input state register, and B labels the registers that store the results of the parallel phase estimations.

We use the circuit $D(U)$ to check if an input state has high overlap with the low-valued eigenphase-space of U [29, 14, 32]. To characterize the low phase space of a unitary U, let $P_\Theta(U)$ (or just P_Θ when U is clear from context) be the projection onto $\text{span}\{|u\rangle : U|u\rangle = e^{i\theta}|u\rangle \text{ with } |\theta| \leq \Theta\}$ (the eigenspace of U with eigenphases less than Θ). Then the following lemma provides key properties of parallel phase estimation circuit $D(U)$:

▶ **Lemma 11** ([29, 14, 32]). *Let U be a unitary on a Hilbert Space $\mathcal{H}_A$, and let $\Theta, \epsilon > 0$. We call Θ the precision and ϵ the accuracy. Then there is a circuit $D(U)$ that acts on the space $\mathcal{H}_A \otimes ((\mathbb{C}^2)^{\otimes b})_B$ for $b = O\left(\log \frac{1}{\Theta} \log \frac{1}{\epsilon}\right)$, and that uses $O\left(\frac{1}{\Theta} \log \frac{1}{\epsilon}\right)$ controlled calls to U. Then for any state $|\psi\rangle \in \mathcal{H}_A$,*

1. $D(U)(P_0|\psi\rangle)_A|0\rangle_B = (P_0|\psi\rangle)_A|0\rangle_B$
2. $\|P_0|\psi\rangle\|^2 \leq \|(I_A \otimes |0\rangle\langle 0|_B)D(U)(|\psi\rangle_A|0\rangle_B)\|^2 \leq \|P_\Theta|\psi\rangle\|^2 + \epsilon$.

Iterative Quantum Amplitude Estimation is a robust version of amplitude estimation, which uses repeated applications of amplitude estimation to achieve improved error bounds:

▶ **Lemma 12** (Iterative Quantum Amplitude Estimation [20]). *Let $\delta > 0$ and $\mathcal{A}$ be a unitary quantum circuit such that on a state $|0\rangle$, $\mathcal{A}|\psi\rangle = \alpha_0|0\rangle|\psi_0\rangle + \alpha_1|1\rangle|\psi_1\rangle$. Then there is an algorithm that estimates $|\alpha_0|^2$ to additive error δ with success probability at least $1 - p$ using $O\left(\frac{1}{\delta} \log\left(\frac{1}{p} \log \frac{1}{\delta}\right)\right)$ calls to $\mathcal{A}$ and $\mathcal{A}^\dagger$.*

A key mathematical tool in analyzing span program algorithms is the Effective Spectral Gap Lemma:

▶ **Lemma 13** (Effective Spectral Gap Lemma, [31]). *Let Π and Λ be projections, and let $U = (2\Pi - I)(2\Lambda - I)$ be the unitary that is the product of their associated reflections. If $\Lambda|w\rangle = 0$, then $\|P_\Theta(U)\Pi|w\rangle\| \leq \frac{\Theta}{2}\||w\rangle\|$.*

We will need the following relationship between optimal positive witnesses and optimal negative approximate witnesses:

▶ **Theorem 14.** [21, Theorem 2.11] *Given a span program $\mathcal{P} = (H, \mathcal{V}, |\tau\rangle, A)$ on R^m and $x \in R^m$, if $|w\rangle$ is the optimal positive witness for x and $\langle\widetilde{\omega}|$ is an optimal negative approximate witness for x, then*

$$|w\rangle = w_+(x)\Pi_{H(x)}(\langle\widetilde{\omega}|A)^\dagger. \tag{11}$$

As discussed following Theorem 10, we decompose the state $|\hat{0}\rangle$ into a linear combination of two orthogonal states. They are

$$\begin{aligned}|\psi_{x,0}\rangle =&|\hat{0}\rangle + \frac{1}{\alpha}|w\rangle,\\ |\psi_{x,+}\rangle =&|\hat{0}\rangle - \frac{\alpha}{w_+(x)}|w\rangle,\end{aligned} \tag{12}$$

so we can write $|\hat{0}\rangle$ as

$$|\hat{0}\rangle = a_0|\psi_{x,0}\rangle + a_+|\psi_{x,+}\rangle, \quad \text{where} \quad a_0 = \frac{1}{1+\frac{w_+(x)}{\alpha^2}}, \qquad a_+ = \frac{1}{1+\frac{\alpha^2}{w_+(x)}}. \tag{13}$$

We first show that $|\psi_{x,0}\rangle$ is a 0-phase eigenvector of $U(\mathcal{P},x,\alpha)$. Note that $\tilde{A}^\alpha|\psi_{x,0}\rangle = \frac{1}{\alpha}(|\tau\rangle - |\tau\rangle) = 0$ (see Equation (10)), so recalling that Λ^α is the orthogonal projector onto the kernel of $\tilde{A}^\alpha$, we have $\Lambda^\alpha|\psi_{x,0}\rangle = |\psi_{x,0}\rangle$. Furthermore, since Π_x is the orthogonal projector onto $\tilde{H}(x) = H(x) \oplus \text{span}\{|\hat{0}\rangle\}$, it follows that $\Pi_x|\psi_{x,0}\rangle = |\psi_{x,0}\rangle$, where we use that $|w\rangle$ is a positive witness, so $|w\rangle \in H(x)$. Thus $U(\mathcal{P},x,\alpha)|\psi_{x,0}\rangle = |\psi_{x,0}\rangle$.

On the other hand $|\psi_{x,+}\rangle$ has low overlap with $P_\Theta(U(\mathcal{P},x,\alpha))$ for small enough Θ and α, as the following lemma shows.

▶ **Lemma 15.** *If $\alpha^2 \geq 1/\widetilde{W}_-$, then $\|P_\Theta(U(\mathcal{P},x,\alpha))|\psi_{x,+}\rangle\| \leq \Theta\alpha\sqrt{\widetilde{W}_-}$.*

Proof. Let $\langle\widetilde{\omega}|$ be an optimal negative approximate witness for x (see Definition 3), and let

$$|v\rangle = |\hat{0}\rangle - \alpha(\langle\widetilde{\omega}|A)^\dagger. \tag{14}$$

Using Theorem 14 and the fact that $\Pi_x|\hat{0}\rangle = |\hat{0}\rangle$, we have that

$$\Pi_x|v\rangle = |\hat{0}\rangle - \alpha\Pi_{H(x)}(\langle\widetilde{\omega}|A)^\dagger = |\hat{0}\rangle - \alpha\frac{|w\rangle}{w_+(x)} = |\psi_{x,+}\rangle. \tag{15}$$

Now we will show $\Lambda^\alpha|v\rangle = 0$. Let $|k\rangle$ be in the kernel of $\tilde{A}^\alpha$, so $\tilde{A}^\alpha|k\rangle = 0$. Using Equation (10) and rearranging,

$$A|k\rangle = \frac{1}{\alpha}|\tau\rangle\langle\hat{0}|k\rangle. \tag{16}$$

Then

$$\begin{aligned}\langle v|k\rangle &= \langle\hat{0}|k\rangle - \alpha\langle\widetilde{\omega}|A|k\rangle\\ &= \langle\hat{0}|k\rangle - \langle\hat{0}|k\rangle\langle\widetilde{\omega}|\tau\rangle\\ &= 0\end{aligned} \tag{17}$$

where we have used Equations (14) and (16) and the properties of optimal negative approximate witnesses. Thus $|v\rangle$ is orthogonal to any element of the kernel of $\tilde{A}^\alpha$, so $\Lambda^\alpha|v\rangle = 0$.

Now we can apply Lemma 13 to $|v\rangle$ to get:

$$\begin{aligned}\|P_\Theta(U(\mathcal{P},x,\alpha))|\psi_{x,+}\rangle\| &= \|P_\Theta(U(\mathcal{P},x,\alpha))\Pi_x|v\rangle\|\\ &\leq \frac{\Theta}{2}\||v\rangle\|\\ &= \frac{\Theta}{2}\sqrt{1+\alpha^2\widetilde{w}_-(x,\mathcal{P})}\\ &\leq \Theta\alpha\sqrt{\widetilde{W}_-},\end{aligned} \tag{18}$$

where in the first line we have used Equation (15), and in the last, our assumption that $\alpha^2\widetilde{W}_- \geq 1$. ◀

▶ **Corollary 16.** $\|P_0(U(\mathcal{P},x,\alpha))|\psi_{x,+}\rangle\| = 0$.

Proof. Apply Lemma 15 with Θ set to 0. ◀

To prove Theorem 10, we analyze the following algorithm:

Algorithm 1 `WitnessGeneration`$(\mathcal{P}, O_x, \delta, \epsilon)$.

Input : Error tolerance δ, accuracy ϵ, span program $\mathcal{P}$ that decides a function f, oracle O_x

Output : A quantum state $|\hat{w}\rangle/\||\hat{w}\rangle\|$ such that for the optimal positive witness $|w\rangle$ for x, $\||w\rangle/\sqrt{w_+(x)} - |\hat{w}\rangle/\||\hat{w}\rangle\|\|^2 \leq O(\epsilon)$ with probability $1 - O(\delta)$

$\epsilon' \leftarrow \min\{\epsilon, 1/96\}$; $\quad T \leftarrow \left\lceil \log \sqrt{W_+ \widetilde{W}_-} \right\rceil$;

$p \leftarrow \min\left\{ \delta/\log(W_+\widetilde{W}_-), 1/\sqrt{W_+\widetilde{W}_-} \right\}$

`// Probing Stage`

for $i = 0$ **to** T **do**

  $\alpha \leftarrow 2^i/\sqrt{\widetilde{W}_-}$

  $\hat{a} \leftarrow$ Iterative Amplitude Estimation (Lemma 12) estimate (with probability of failure p and additive error $1/48$) of the probability of outcome $|0\rangle_B$ in register B when $D(U(\mathcal{P},x,\alpha))$ (see Lemma 11) acts on $|\hat{0}\rangle_A|0\rangle_B$ with error ϵ', precision $\sqrt{\frac{\epsilon'}{\alpha^2\widetilde{W}_-}}$

  if $\frac{15}{48} \leq \hat{a} \leq \frac{35}{48}$ **then** Break

`// State Generation Stage`

for $j = 1$ **to** $\log(1/\delta)$ **do**

  Apply $D(U(\mathcal{P},x,\alpha))$ to $|\hat{0}\rangle_A|0\rangle_B$ with error ϵ', precision $\sqrt{\frac{\epsilon'}{\alpha^2\widetilde{W}_-}}$

  Make a measurement with outcome $M = \{(I - |\hat{0}\rangle\langle\hat{0}|)_A \otimes |0\rangle\langle 0|_B\}$ on the resultant state

  if *Measure outcome* M **then**

    Return the resultant state

Return "failure"

To analyze Algorithm 1, will need the following lemma and corollary. In Algorithm 1, we estimate the probability of measuring the outcome $|0\rangle$ in the B register after doing phase estimation. In the following lemma, we prove this probability is closely related to a_0 from Equation (13).

▶ **Lemma 17.** *Applying $D(U(\mathcal{P},x,\alpha)))$ with error ϵ and precision $\sqrt{\frac{\epsilon}{\alpha^2\widetilde{W}_-}}$ (see Lemma 11) to input state $|\hat{0}\rangle_A|0\rangle_B$ for $\alpha \geq 1/\sqrt{\widetilde{W}_-}$ results in the outcome $|0\rangle$ in the B register with probability in the range $[a_0, a_0 + 2\epsilon]$.*

Proof. Throughout the proof, let $U = U(\mathcal{P},x,\alpha)$. The probability that we measure $|0\rangle$ in register B after we apply $D(U)$ with error ϵ and precision Θ to $|\hat{0}\rangle_A|0\rangle_B$ is, by Lemma 11 Item 2, at most

$$\|P_\Theta(U)|\hat{0}\rangle\|^2 + \epsilon = \|a_0 P_\Theta(U)|\psi_{x,0}\rangle + a_+ P_\Theta(U)|\psi_{x,+}\rangle\|^2 + \epsilon, \tag{19}$$

by Equation (13). Now $P_\Theta(U)|\psi_{x,0}\rangle$ and $P_\Theta(U)|\psi_{x,+}\rangle$ are orthogonal, since

$$\langle\psi_{x,0}|P_\Theta(U)P_\Theta(U)|\psi_{x,+}\rangle = \langle\psi_{x,0}|\psi_{x,+}\rangle = 0, \tag{20}$$

where we've used that $P_\Theta(U)|\psi_{x,0}\rangle = |\psi_{x,0}\rangle$ and that $|\psi_{x,0}\rangle$ and $|\psi_{x,+}\rangle$ are orthogonal. Continuing from Equation (19) and using the orthogonality condition, we have, using $\Theta = \sqrt{\frac{\epsilon}{\alpha^2 \widetilde{W}_-}}$,

$$\begin{aligned}
\|P_\Theta(U)|\hat{0}\rangle\|^2 + \epsilon &= a_0^2 \|P_\Theta(U)|\psi_{x,0}\rangle\|^2 + a_+^2 \|P_\Theta(U)|\psi_{x,+}\rangle\|^2 + \epsilon \\
&\leq a_0^2 \||\psi_{x,0}\rangle\|^2 + a_+^2 \Theta^2 \alpha^2 \widetilde{W}_- + \epsilon \qquad \text{by Lemma 15, since } \alpha^2 \widetilde{W}_- \geq 1 \\
&\leq a_0 + a_+^2 \epsilon + \epsilon \\
&\leq a_0 + 2\epsilon, \qquad (21)
\end{aligned}$$

where we have used that $\||\psi_{x,0}\rangle\|^2 = 1/a_0$, and $a_+ \leq 1$ (see Equation (13)) .

By Lemma 11 Item 2, the probability that we measure $|0\rangle$ in register B after applying $D(U(\mathcal{P}, x, \alpha))$ on $|\hat{0}\rangle_A|0\rangle_B$ with error ϵ and any precision is at least

$$\|P_0(U)|\hat{0}\rangle\|^2 = \|a_0 P_0(U)|\psi_{x,0}\rangle + a_+ P_0(U)|\psi_{x,+}\rangle\|^2 = a_0^2 \||\psi_{x,0}\rangle\|^2 = a_0, \qquad (22)$$

where we have used Corollary 16. ◀

▶ **Corollary 18.** *In Algorithm 1, if in an iteration of the Probing Stage, Iterative Amplitude Estimation does not fail at Line 4 and subsequently causes a break at Line 5, then*

$$a_0 \in \left[\frac{1}{4}, \frac{3}{4}\right], \qquad \frac{a_0^2 w_+(x)}{\alpha^2} \in \left[\frac{3}{16}, \frac{1}{4}\right]. \qquad (23)$$

Proof. If Iterative Amplitude Estimation does not fail at Line 4 and causes a break at Line 5, then we have an estimate $\hat{a}$ that is in the range $[\frac{15}{48}, \frac{35}{48}]$. Thus, because of the additive error of $1/48$ in Iterative Amplitude Estimation, the probability of measuring outcome $|0\rangle_B$ is in the range $[\frac{14}{48}, \frac{36}{48}]$. By Lemma 17, this same probability is in the range $[a_0, a_0 + 2\epsilon']$, so in particular these two ranges overlap. Thus, since we choose $2\epsilon'$ to be at most $1/48$, we have that

$$a_0 \in \left[\frac{13}{48}, \frac{36}{48}\right] \subset \left[\frac{1}{4}, \frac{3}{4}\right]. \qquad (24)$$

Using $a_0 = (1 + \frac{w_+(x)}{\alpha^2})^{-1}$ (see Equation (13)), this implies the stated ranges for $\frac{a_0^2 w_+(x)}{\alpha^2} = a_0(1 - a_0)$. ◀

Now we prove the main performance guarantees of Algorithm 1, bounding the success probability and the expected query complexity, thus proving Theorem 10.

Proof of Theorem 10. Letting $U = U(\mathcal{P}, x, \alpha)$, we analyze Algorithm 1. We first show that the algorithm will produce the desired state if both the Probing Stage and the State Generation stage are successful. Then we will analyze the probability of this occurring, in order to bound the success probability of the algorithm.

We say the Probing Stage is successful if in some iteration, Iterative Amplitude estimation, having not failed thus far, does not fail and then triggers a break at Line 6, in which case we can apply Corollary 18. Under these assumptions, we consider the outcome of a successful State Generation stage, when we achieve the measurement outcome $M = (I - |\hat{0}\rangle\langle\hat{0}|)_A \otimes |0\rangle\langle 0|_B$. The non-normalized state $|\hat{w}\rangle$ that is produced upon measurement outcome M is

$$
\begin{aligned}
|\hat{w}\rangle &= (I - |\hat{0}\rangle\langle\hat{0}|)_A \otimes |0\rangle\langle 0|_B D(U)|\hat{0}\rangle_A|0\rangle_B \\
&= a_0(I - |\hat{0}\rangle\langle\hat{0}|)_A D(U)|\psi_{x,0}\rangle_A|0\rangle_B + a_+(I - |\hat{0}\rangle\langle\hat{0}|)_A \otimes |0\rangle\langle 0|_B D(U)|\psi_{x,+}\rangle_A|0\rangle_B \\
&= a_0(I - |\hat{0}\rangle\langle\hat{0}|)_A|\psi_{x,0}\rangle_A|0\rangle_B + a_+(I - |\hat{0}\rangle\langle\hat{0}|)_A \otimes |0\rangle\langle 0|_B D(U)|\psi_{x,+}\rangle_A|0\rangle_B \\
&= \frac{a_0}{\alpha}|w\rangle_A|0\rangle_B + \underbrace{a_+(I - |\hat{0}\rangle\langle\hat{0}|)_A \otimes |0\rangle\langle 0|_B D(U)|\psi_{x,+}\rangle_A|0\rangle_B}_{=:|\xi\rangle}, \qquad (25)
\end{aligned}
$$

where in the final equality, we used Lemma 11 Item 1, since $P_0(U)|\psi_{x,0}\rangle = |\psi_{x,0}\rangle$.

We would like to bound Δ, where

$$
\begin{aligned}
\Delta := \left\| \frac{|\hat{w}\rangle}{\||\hat{w}\rangle\|} - \frac{|w\rangle_A|0\rangle_B}{\sqrt{w_+(x)}} \right\| &= \left\| \frac{\frac{a_0}{\alpha}|w\rangle_A|0\rangle_B + |\xi\rangle}{\||\hat{w}\rangle\|} - \frac{|w\rangle_A|0\rangle_B}{\sqrt{w_+(x)}} \right\| \\
&\leq \left| \frac{a_0}{\alpha\,\||\hat{w}\rangle\|} - \frac{1}{\sqrt{w_+(x)}} \right| \||w\rangle\| + \frac{\||\xi\rangle\|}{\||\hat{w}\rangle\|} \quad \text{by triangle ineq.} \\
&\leq \left| \frac{a_0\sqrt{w_+(x)}}{\alpha\,\||\hat{w}\rangle\|} - 1 \right| + \frac{\||\xi\rangle\|}{\||\hat{w}\rangle\|}. \qquad (26)
\end{aligned}
$$

To bound $\||\xi\rangle\|$, we have

$$
\begin{aligned}
\||\xi\rangle\|^2 = a_+^2 \left\|(I - |\hat{0}\rangle\langle\hat{0}|)_A \otimes |0\rangle\langle 0|_B D(U)|\psi_{x,+}\rangle_A|0\rangle_B\right\|^2 &\leq \|I_A \otimes |0\rangle\langle 0|_B D(U)|\psi_{x,+}\rangle_A|0\rangle_B\|^2 \\
&\leq \|P_\Theta|\psi_{x,+}\rangle\|^2 + \epsilon' \\
&\leq \Theta^2\alpha^2\widetilde{W}_- + \epsilon' \leq 2\epsilon', \qquad (27)
\end{aligned}
$$

where the first inequality is because a projection can only decrease the norm of a vector, and $a_+ \leq 1$; the second inequality is from by Lemma 11 Item 2, and the third inequality comes from Lemma 15 and our choice of Θ.

Next, to bound $\||\hat{w}\rangle\|$, we use the triangle inequality on the final line of Equation (25), and Equation (27) to get

$$
\frac{a_0\sqrt{w_+(x)}}{\alpha} - \sqrt{2\epsilon'} \leq \||\hat{w}\rangle\| \leq \frac{a_0\sqrt{w_+(x)}}{\alpha} + \sqrt{2\epsilon'}. \qquad (28)
$$

By our choice of ϵ', we have $2\epsilon' \leq 1/48$, and also applying Corollary 18 to Equation (28), we have

$$
\frac{1}{4} < \sqrt{3/16} - \sqrt{1/48} \leq \||\hat{w}\rangle\| \leq \sqrt{1/4} + \sqrt{1/48} < \frac{3}{4}. \qquad (29)
$$

Rearranging Equation (28) and applying Equation (29), we have

$$
\left| \frac{a_0\sqrt{w_+(x)}}{\alpha\,\||\hat{w}\rangle\|} - 1 \right| \leq \frac{\sqrt{2\epsilon'}}{\||\hat{w}\rangle\|}. \qquad (30)
$$

Then plugging Equations (27), (29), and (30) into Equation (26) we have:

$$
\Delta \leq \frac{2\sqrt{2\epsilon'}}{\||\hat{w}\rangle\|} < 8\sqrt{2\epsilon'} = O(\epsilon). \qquad (31)
$$

Now we analyze the probability that both the Probing Stage and State Generation Stage are successful, resulting in the state $|\hat{w}\rangle/\||\hat{w}\rangle\|$ as in Equation (26). First note that there is a value of α (if we iterate in the Probing Stage long enough), that will cause us to break out of the Probing Stage if Iterative Amplitude Estimation does not fail. In particular, when $w_+(x)/\alpha^2 \in [1/2, 2]$, then from Equation (13) $a_0 \in [1/3, 2/3]$. Thus by Lemma 17 and since $2\epsilon' \leq 1/48$, the probability of of outcome $|0\rangle_B$ is in $[16/48, 33/48]$, which in Line 5 causes us to leave the Probing Stage if Iterative Amplitude Estimation does not fail. This occurs for some value of α, as we are doubling α at each iteration of the Probing Stage, causing $w_+(x)/\alpha^2$ to decrease, and initially we have $w_+(x)/\alpha^2 = w_+(x)\widetilde{W}_- \geq 1$.[5]

Thus if no error occurs, the condition of Line 5 will be satisfied after some number L of rounds such that $L \in O(\log(w_+(x)\widetilde{W}_-)) = O(\log(W_+\widetilde{W}_-))$. As the probability of failing a single Iterative Amplitude Estimation round is $p \leq \delta/\log(W_+\widetilde{W}_-)$ (see Line 1), the probability of leaving the Probing Stage when Line 5 is satisfied (rather than before or after) is at least

$$(1-p)^L = 1 - O(\delta). \tag{32}$$

Assuming that we have successfully left the Probing Stage without failure, we next calculate the probability of getting a measurement outcome M during the at most $\log(1/\delta)$ iterations of the State Generation Stage. The probability of getting outcome M is lower bounded by (from Equation (29))

$$\||\hat{w}\rangle\|^2 \geq 1/16. \tag{33}$$

Thus the probability of success in the State Generation Stage is

$$1 - (15/16)^{\log(1/\delta)} = 1 - O(\delta). \tag{34}$$

Combining Equations (32) and (34), our probability of successfully producing a state $|\hat{w}\rangle/\||\hat{w}\rangle\|$ as in Equation (26) is

$$(1 - O(\delta))(1 - O(\delta)) = 1 - O(\delta). \tag{35}$$

To calculate the expected query complexity, we first note that if we terminate in round $t \in \{0, \ldots, \lceil \log \sqrt{W_+} \rceil\}$ of the Probing Stage, we use

$$\begin{aligned} &\sum_{i=0}^{t} O\left(\frac{2^i}{\sqrt{\epsilon}} \log\left(\frac{1}{\epsilon}\right) \log\left(\frac{1}{p}\right)\right) + O\left(\log\left(\frac{1}{\delta}\right) \frac{2^t}{\sqrt{\epsilon}} \log\left(\frac{1}{\epsilon}\right)\right) \\ =&O\left(\frac{2^t}{\sqrt{\epsilon}} \log\left(\frac{1}{\epsilon}\right) \log\left(\frac{1}{p\delta}\right)\right) \end{aligned} \tag{36}$$

queries, which comes from the cost of Iterative Amplitude Estimation (Lemma 12) applied to phase estimation (Lemma 11) in each round of the Probing Stage up to the t^{th} round, plus the cost of phase estimation in the State Conversion Stage.

The probability that we terminate in any round t when we have an estimate $\hat{a}$ that is not in the range $[\frac{15}{48}, \frac{35}{48}]$ is at most p. Using Equation (36) the the total contribution to the average query complexity from all such rounds is at most

$$\sum_{t=0}^{\lceil \log \sqrt{W_+\widetilde{W}_-} \rceil} O\left(p\frac{2^t}{\sqrt{\epsilon}} \log\left(\frac{1}{\epsilon}\right) \log\left(\frac{1}{p\delta}\right)\right) = O\left(p\sqrt{\frac{W_+\widetilde{W}_-}{\epsilon}} \log\left(\frac{1}{\epsilon}\right) \log\left(\frac{1}{p\delta}\right)\right). \tag{37}$$

[5] To see that $w_+(x)\widetilde{W}_- \geq 1$, let $N_+ = \min\{\||w\rangle\|^2 : A|w\rangle = |\tau\rangle\}$, and $N_- = \min\{\|\langle\omega|A\|^2 : \langle\omega|\tau\rangle = 1\}$. Then $w_+(x) \geq N_+$, and $\widetilde{W}_- \geq N_-$, and by [21, Section 2.4], $N_+N_- = 1$.

where in the sum we have actually included all rounds, not just those that satisfy when $\hat{a}$ is not in the range $[\frac{15}{48}, \frac{35}{48}]$, which is acceptable since we are deriving an upper bound on the expected query complexity.

If we terminate at a round t^* when $\hat{a}$ is in the range $[\frac{15}{48}, \frac{35}{48}]$, which happens when Iterative Amplitude Estimation does not fail at Line 4 and then causes a break at Line 5, from Equation (23) we have $\frac{w_+(x)}{\alpha^2} \in \left[\frac{1}{3}, 4\right]$, and $2^{t^*} = \alpha\sqrt{\widetilde{W}_-}$ so $\sqrt{w_+(x)\widetilde{W}_-}/2 \leq 2^{t^*} \leq \sqrt{3w_+(x)\widetilde{W}_-}$. Because we double α at each iteration, there are only a constant number of rounds where we will find $\hat{a}$ in the appropriate range, and we trivially upper bound the probability of terminating at any such round by 1. Using Equation (36), these rounds add

$$O\left(\sqrt{\frac{w_+(x)\widetilde{W}_-}{\epsilon}}\log\left(\frac{1}{\epsilon}\right)\log\left(\frac{1}{p\delta}\right)\right) \tag{38}$$

to the total expected query complexity.

Combining Equations (37) and (38), and using that we set p to be $O\left(1/\sqrt{W_+\widetilde{W}_-}\right)$ (Line 1), we find the expected query complexity is

$$O\left(\sqrt{\frac{w_+(x)\widetilde{W}_-}{\epsilon}}\log\left(\frac{1}{\epsilon}\right)\log\left(\frac{1}{p\delta}\right)\right) = \widetilde{O}\left(\sqrt{\frac{w_+(x)\widetilde{W}_-}{\epsilon}}\log\left(\frac{1}{\delta}\right)\right). \tag{39}$$

4 Graph Applications

4.1 Finding an Edge on a Path

In this section, we consider the problem of finding an edge on an st-path in $G(x)$, which we denote st-EDGE$_G(x)$. That is, given query access to a string x that determines a subgraph $G(x) = (V, E(x))$ of an n-vertex graph G, as described in Section 2.3 (if G is a complete graph, x is just the adjacency matrix of $G(x)$), with $s, t \in V$ such that there is at least one path from s to t in $G(x)$, output an edge $e \in E(x)$ that is on a (self-avoiding) path from s to t.

Classically, it is hard to imagine that this problem is much easier than finding a path, and indeed, in our classical lower bound in Theorem 20 the set-up forces the algorithm to learn a complete path before it can find any edge on the path. However, we find that quantumly, when there are short or multiple paths, this problem is easier than any path finding algorithms known. This opens up the possibility of improved quantum algorithms for cases where it is not necessary to know the complete path, like the st-cut set algorithm of Section 4.2.

▶ **Theorem 19.** *Fix $p > 0$, and a family of n-vertex simple graphs G with vertices s and t. There is a quantum algorithm (Algorithm 2) that solves st-EDGE$_G(x)$ with probability $1-O(p)$ and uses $\widetilde{O}\left(\frac{n\sqrt{R_{s,t}(G(x))}}{p}\right)$ expected queries on input x. More precisely, with probability $1-O(p)$, the algorithm samples from a distribution $\hat{q}$ such the total variation distance between $\hat{q}$ and $q_{G(x),s,t}$ is $O(\sqrt{p})$, where $q_{G(x),s,t}(u,v)$ (defined in Equation (7)) is proportional to $\theta^*(u,v)^2$, where θ^* is the optimal unit st-flow on $G(x)$.*

To obtain this result, we run our witness state generation algorithm (Algorithm 1) using the span program for st-connectivity, $\mathcal{P}_{G_{st}}$ and an oracle O_x that defines a graph $G(x)$ with a path between s and t. When successful, the output will be a quantum state that is approximately proportional to the optimal flow state, Equation (6), which itself is a superposition of edges on paths by Lemma 7. Then from Equation (7), when we then measure in the standard basis, the probability of obtaining an edge e should be close to $q_{G(x),s,t}(e)$, and with high probability, we will measure some edge on a path.

Proof of Theorem 19: We analyze Algorithm 2.

Algorithm 2 `EdgeFinder`(O_x, p, G, s, t).

Input : Failure tolerance $p > 0$, oracle O_x for the graph $G(x) = (V, E(x))$, $s, t \in V$ such that there is a path from s to t.

Output : An output e, or "Failure", such that with probability $1 - O(p)$, e is an edge on a path from s to t.

$\epsilon \leftarrow p^2$; $\delta \leftarrow p$

$|\hat{\theta}\rangle \leftarrow$ `WitnessGeneration`$(\mathcal{P}_{G_{st}}, O_x, \epsilon, \delta)$ (Algorithm 1)

if $|\hat{\theta}\rangle \neq$ *"Failure"* **then**

  $e \leftarrow$result of Measuring $|\hat{\theta}\rangle$ in the standard basis

Return "Failure"

If `WitnessGeneration`$(\mathcal{P}_{G_{st}}, O_x, \epsilon, \delta)$ (see Algorithm 1) does not fail, which happens with probability $1 - O(\delta) = 1 - O(p)$, then by Theorem 10,

$$|\hat{\theta}\rangle = |\theta^*\rangle / \||\theta^*\rangle\| + |\eta\rangle \tag{40}$$

for some $|\eta\rangle$ such that $\||\eta\rangle\|^2 = O(\epsilon)$ and from Equation (6), $|\theta^*\rangle = \frac{1}{2}\sum_{e\in\overrightarrow{E}} \theta^*(e)|e\rangle$ where θ^* is the optimal unit st-flow in $G(x)$, so $\||\theta^*\rangle\| = \sqrt{\mathcal{R}_{s,t}(G(x))}$.

Let $P_{E(x),s,t}$ be the projection onto the set of edges in $\overrightarrow{E}(x)$ that are on (self-avoiding) paths from s to t. The probability that we measure such an edge when we measure $|\hat{\theta}\rangle$ in the standard basis is the square of

$$\left\|P_{E(x),s,t}|\hat{\theta}\rangle\right\| \geq \left\|P_{E(x),s,t}|\theta^*\rangle / \||\theta^*\rangle\|\right\| - \left\|P_{E(x),s,t}|\eta\rangle\right\| = 1 - O(\sqrt{\epsilon}), \tag{41}$$

where we have used the triangle inequality, and the fact that $P_{E(x),s,t}|\theta^*\rangle = |\theta^*\rangle$, by Lemma 7. Continuing, we have probability

$$\left\|P_{E(x),s,t}|\hat{\theta}\rangle\right\|^2 \geq \left(1 - O(\sqrt{\epsilon})\right)^2 = 1 - O(\sqrt{\epsilon}). \tag{42}$$

Thus our total probability of success of measuring an edge on a path is $(1-O(\delta))(1-O(\sqrt{\epsilon}))$. Since we are setting ϵ to p^2 and δ to p, our total probability of success is $1 - O(p)$.

Let $\hat{q}$ be the output distribution of Algorithm 2. By the relationship between total variation distance and trace norm, we have that $d(\hat{q}, q_{G(x),s,t})$, the total variation distance between $\hat{q}$ and $q_{G(x),s,t}$, is at most the trace norm of $|\hat{\theta}\rangle$ and $|\theta^*\rangle/\||\theta^*\rangle\|$ (see e.g. [33]) so

$$\begin{aligned} d(\hat{q}, q_{G(x),s,t}) &\leq \sqrt{1 - \left|\langle\hat{\theta}|\theta^*\rangle / \||\theta^*\rangle\|\right|^2} \\ &= \sqrt{1 - \left|\langle\hat{\theta}|\hat{\theta}\rangle - \langle\hat{\theta}|\eta\rangle\right|^2} \\ &\leq \sqrt{1 - (1 - \||\eta\rangle\|)^2} \\ &\leq \sqrt{2\,\||\eta\rangle\|} = O(\epsilon^{1/4}) = O(\sqrt{p}). \end{aligned} \tag{43}$$

By Theorem 10, the expected query complexity of `WitnessGeneration`, and thus Algorithm 2 is

$$\widetilde{O}\left(\sqrt{\frac{w_+(x)\widetilde{W}_-}{\epsilon}}\log\left(\frac{1}{\delta}\right)\right) = \widetilde{O}\left(\frac{\sqrt{R_{s,t}(G(x))n}}{p}\right) \tag{44}$$

where we have used the fact that, for $\mathcal{P}_{G_{st}}$, $w_+(x) = R_{s,t}(G(x))$ and $\widetilde{W}_- = O(n^2)$ [7, 21]; and set ϵ to p^2 and δ to p, as in Algorithm 2. ◀

We can use Theorem 19 to prove the following separation between the quantum and classical query complexity of finding an edge on a path:

▶ **Theorem 20.** *Let $G = (V, E)$ with $s, t \in V$ be an n-vertex complete graph, and suppose we are promised that $G(x)$ has a path of length L for $L \in [3, n/4]$ between s and t (L may depend on x and need not be known ahead of time). Then st-EDGE$_G(x)$ can be solved in $\widetilde{O}(n\sqrt{L})$ expected quantum queries on input x, while any classical algorithm has query complexity $\Omega(n^2)$.*

Proof. For the quantum algorithm, we apply Theorem 19 with bounded probability of error $p = \Omega(1)$, and use the fact that $R_{s,t}(G) = O(L)$.

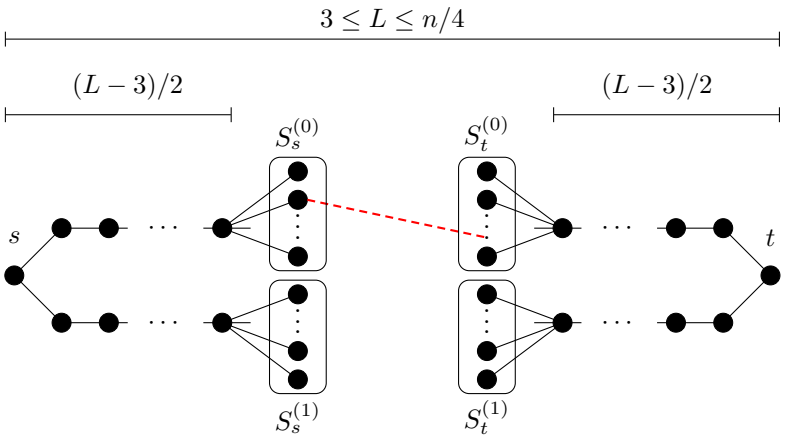

Figure 1 The solid black lines show the edges that are present in $G(x)$ for any x. In addition, $G(x)$ contains a single edge between a vertex in $S_s^{(b)}$ and $S_t^{(b)}$, where $b = \sigma_1^*$, as in the dashed red edge, resulting in a single path of length L.

For the classical lower bound, we reduce the following problem to path edge finding: Given a string x of $N = 2^\ell$ bits, $(x_\sigma)_{\sigma\in\{0,1\}^\ell}$ such that there is a unique σ^* with $x_{\sigma^*} = 1$, output σ_1^*. That is, we would like to output the first bit of the index of the unique 1-valued bit of x. By an adversary argument similar to a standard OR lower bound, the bounded error randomized query complexity of this problem is $\Omega(N)$. We will show how to solve this problem with an algorithm for finding a path edge on a graph like the one depicted in Figure 1.

For $x \in \{0,1\}^N$, let $G(x)$ be a graph on $n = \Theta(2^{\ell/2})$ vertices in which there is a unique st-path of length L, for some odd L, as shown in Figure 1. The vertex s is connected by a path of length $(L-3)/2$ to a vertex that is additionally connected to a set of $2^{(\ell-1)/2}$ vertices, $S_s^{(0)} = \{u_{0,\sigma} : \sigma \in \{0,1\}^{(\ell-1)/2}\}$. In a symmetric manner, s is also connected by another disjoint path of length $(L-3)/2$ to a vertex that is additionally connected to a

set of $2^{(\ell-1)/2}$ vertices, $S_s^{(1)} = \{u_{1,\sigma} : \sigma \in \{0,1\}^{(\ell-1)/2}\}$. In the same way, t is connected by a pair of disjoint paths of length $(L-3)/2$ to a pair of vertices, additionally connected to $S_t^{(0)} = \{v_{0,\sigma} : \sigma \in \{0,1\}^{(\ell-1)/2}\}$ and $S_t^{(1)} = \{v_{1,\sigma} : \sigma \in \{0,1\}^{(\ell-1)/2}\}$ respectively. All edges described so far (the black edges in Figure 1) are always present in $G(x)$ (we simulate querying the associated input bits by just outputting 1). We now describe edges whose presence in $G(x)$ is determined by x. For $b \in \{0,1\}$, there is a potential edge between every pair of vertices $u_{b,\sigma} \in S_s^{(b)}$ and $v_{b,\sigma'} \in S_t^{(b)}$, with the label $x_{b\sigma\sigma'}$, meaning exactly one of these is present in $G(x)$ – the one with $\sigma^* = b\sigma\sigma'$. All remaining possible edges are never present in $G(x)$ (we simulate querying their associated input bits by just outputting 0).

We can find the first bit of σ^* by running the edge finding algorithm on $G(x)$. Assuming the output is correct, there are the following possibilities:

1. If the algorithm outputs an edge from the middle part of the graph, then it must be the one labelled by x_{σ^*}, so σ^* is learned entirely.
2. If the algorithm outputs an edge from the left-hand side of the graph, it is on a path between s and $S_s^{(b)}$ for some $b \in \{0,1\}$, and we know that $\sigma_1^* = b$.
3. If the algorithm outputs an edge from the right-hand side of the graph, it is on a path between t and $S_t^{(b)}$ for some $b \in \{0,1\}$, and we know that $\sigma_1^* = b$.

In all cases, we have learned σ_1^*. This gives a lower bound on path-edge finding of $\Omega(N) = \Omega(2^\ell) = \Omega(n^2)$. ◀

4.2 Finding an *st*-cut set

Given a graph $G(x)$ containing a path from s to t, an *st*-cut set is a set of edges in $G(x)$ such that when those edges are removed from $G(x)$, there is no longer a path from s to t. The *st*-cut set problem is that of finding an *st*-cut set. This problem has applications to detecting weak points in networks in order to figure out how to strengthen a network, or conversely, for sabotaging networks.

We first note that for graphs with a single *st*-path, Theorem 19 can immediately be used to find an *st*-cut set, since any edge on the path is an *st*-cut set. However, we can also analyze more complex situations, as the following, in which we have an upper bound on the effective resistance of the graph, and a lower bound on the optimal unit *st*-flow going through any edge in the *st*-cut set:

▶ **Theorem 21.** *For functions $R, g : \mathbb{N} \to \mathbb{R}_{>0}$, let $G = (V, E)$ with $s, t \in V$ be a family of n-vertex simple graphs, and suppose we are additionally promised that $R_{s,t}(G(x)) \leq R(n)$, and there exists an st-cut set $C \subseteq E(x)$ such that for each $\{u, v\} \in C$, $\theta^*(u, v)^2 \geq g(n)$ where θ^* is the optimal unit st-flow in $G(x)$. Then there is a quantum algorithm that outputs a set C' such that $C \subseteq C'$ with bounded error, and has worst-case query complexity $\widetilde{O}\left(\frac{R(n)^2 n}{g(n)^{3/2}}\right)$.*

We can assume without loss of generality that the C in Theorem 21 is a minimal *st*-cut. While we are not guaranteed that the set C' output by the algorithm referred to in Theorem 21 is minimal, it is still an *st*-cut as long as it contains C, since its removal will disconnect s and t.

To prove Theorem 21, we will use the following variation of the well-known "coupon collector" problem.

▶ **Lemma 22.** *Consider repeatedly sampling a random variable Z on a finite set $\mathcal{S}$. Let $C \subseteq \mathcal{S}$ be such that for each $e \in C$, $\Pr[Z = e] \geq B$. Let T be the number of samples to Z before we have sampled each element of C at least once. Then $\mathbb{E}[T] = O\left(\frac{\log |C|}{B}\right)$.*

Proof. For $i \in \{1, \ldots, |C|\}$, the probability that Z is a new element of C, after $i-1$ elements have already been collected, is $p_i \geq (|C| - (i-1))B$. Let T_i be the number of samples to Z after sampling $(i-1)$ elements of C, until we sample i elements of C, so T_i is a geometric random variable with

$$\mathbb{E}[T_i] = 1/p_i \leq ((|C| - (i-1))B)^{-1}. \tag{45}$$

From this we can compute

$$\mathbb{E}[T] = \sum_{i=1}^{|C|} \mathbb{E}[T_i] \leq \sum_{i=1}^{|C|} \frac{1}{(|C| - (i-1))B} = \frac{1}{B} \sum_{j=1}^{|C|} \frac{1}{j} = \Theta\left(\frac{\log |C|}{B}\right). \tag{46}$$

◀

Proof of Theorem 21. We use parameters T' and ϵ, to be defined shortly, and $\delta = 1/4$. Our strategy is to repeatedly run `WitnessGeneration`$(\mathcal{P}_{G_{st}}, O_x, \epsilon, \delta)$ (Algorithm 1) to produce an approximate witness state, and then measure the resultant state in the standard basis to get an edge e, which we add to C'. We repeat this T' times, before outputting C'.

Let Z be the random variable on $E \cup \{\text{Failure}\}$ representing the measured output of one call to Algorithm 1. We set $\epsilon = \Theta\left(\frac{g(n)}{R(n)}\right)$ small enough so that if the algorithm does not fail, we produce a state $|\theta^*\rangle/\||\theta^*\rangle\| + |\eta\rangle$ where $\||\eta\rangle\|^2 \leq g(n)/R(n)$ (see Equation (40) and following discussion). Then the probability that we sample an edge $e' \in C$ when we measure in the standard basis is

$$\begin{aligned} \left\|\langle e'| \left(|\theta^*\rangle/\||\theta^*\rangle\| + |\eta\rangle\right)\right\|^2 &= \|2\theta^*(e')/\sqrt{R_{s,t}(G(x))} - \langle e'|\eta\rangle\|^2 \\ &\geq \|2\sqrt{g(n)/R(n)} - \sqrt{g(n)/R(n)}\|^2 \\ &= \Omega(g(n)/R(n)). \end{aligned} \tag{47}$$

Since the probability of one call to Algorithm 1 not failing is $1 - \delta = \Omega(1)$, for every $e' \in C$, we have $\Pr[Z = e'] \geq B$ for some $B = \Omega(g(n)/R(n))$. Thus, by Lemma 22, the expected number of calls to Algorithm 1 before $C \subseteq C'$ is at most:

$$\mathbb{E}[T] = O\left(\frac{R(n)}{g(n)} \log |C|\right) = O\left(\frac{R(n)}{g(n)} \log n\right). \tag{48}$$

By Markov's inequality, if we set $T' = 100\mathbb{E}[T]$, the algorithm will succeed with bounded error.

By Theorem 10, each call to Algorithm 1 has expected query complexity

$$\widetilde{O}\left(\sqrt{\frac{R_{s,t}(G(x))n^2}{\epsilon}}\right) = \widetilde{O}\left(n\sqrt{\frac{R(n)}{g(n)/R(n)}}\right) = \widetilde{O}\left(\frac{nR(n)}{\sqrt{g(n)}}\right), \tag{49}$$

so the total expected query complexity is

$$\widetilde{O}\left(T'\frac{nR(n)}{\sqrt{g(n)}}\right) = \widetilde{O}\left(\frac{R(n)}{g(n)}\frac{nR(n)}{\sqrt{g(n)}}\right) = \widetilde{O}\left(\frac{nR(n)^2}{g(n)^{3/2}}\right). \tag{50}$$

We can get a worst case algorithm by stopping after 100 times the expected number of steps, if the algorithm is still running, and outputting the current C'. We have no guarantee on the correctness of C' in that case, but by Markov's inequality, this only happens with probability $1/100$. ◀

We can use Theorem 21 to prove the following result for finding an st-cut set in a particular family of graphs with expander subgraphs and a single st-cut edge.

▶ **Corollary 23.** *Let $G = (V, E)$ with $s, t \in V$ be a family of n-vertex graphs, and suppose we are additionally promised that $G(x)$ consists of two disjoint, d-regular (for $d \geq 3$), constant expansion subgraphs, each on $n/2$ vertices, where s and t are always put in separate subgraphs, plus a single additional edge connecting the two subgraphs. Then there is a quantum algorithm that finds the st-cut edge with bounded error in worst-case $\widetilde{O}(n)$ queries, while any classical algorithm has query complexity $\Omega(n^2)$.*

Proof. For a classical algorithm, even if the algorithm had complete knowledge of the two subgraphs, there would be $\Omega(n^2)$ possible locations for the connecting edge, reducing the problem to search, requiring $\Omega(n^2)$ queries.

For the quantum algorithm, note that the maximum effective resistance between any two points in a d-regular (for $d \geq 3$), constant expansion graph on n-vertices is $O(1/d)$ [10]. Thus $R_{s,t}(G(x)) = \Omega(1)$. Additionally, since there is only one edge e' connecting the two subgraphs, the optimal unit st-flow on e', $\theta^*(e')$, must be equal to 1.

Applying Theorem 21 with $R(n) = O(1)$ and $g(n) = \Omega(1)$, we get a worst-case bounded error quantum query complexity $\widetilde{O}(n)$. ◀

4.3 Path Finding

In this section, we consider the problem of finding an st-path in $G(x)$, which we denote st-PATH$_G(x)$. That is, given query access to a string x that determines a subgraph $G(x) = (V, E(x))$ of an n-vertex graph G, as described in Section 2.3 (if G is a complete graph, x is just the adjacency matrix of $G(x)$), with $s, t \in V$ such that there is at least one path from s to t in $G(x)$, output a path from s to t. A path is a sequence of *distinct* vertices $\vec{u} = (u_0, \ldots, u_\ell)$ such that $s = u_0$, $t = u_\ell$, and for all $i \in [\ell]$, $(u_{i-1}, u_i) \in \overrightarrow{E}(G(x))$.

To solve st-PATH$_G$, one might expect that we could simply apply Algorithm 2 multiple times, storing each edge's endpoints and identifying vertices of the endpoints of found edges to reduce the size of the graph, until a path is found. However, such an algorithm could run into challenges that could produce slow running times. For example, in a graph where there are many st-paths, the algorithm could spend too much time sampling edges from different paths, rather than focusing on completing a single path. In the case of a single st-path, such a strategy would not take advantage of the fact that once one edge on the path is found, the problem reduces to two connectivity subproblems (from s to the found edge, and from t to the found edge) that each typically have significantly smaller query complexities than the original problem.

Thus we develop two algorithms that allow us to prove tighter expected query complexity bounds than Ref. [18] for the case of short longest st-paths, one in the case of a single st-path, and one for generic graphs.

Before getting into quantum algorithms for path detection, we note the following corollary of Theorem 20, via a reduction to path finding from path-edge finding, that characterizes the classical query complexity of path finding in the case of short longest st-paths:

▶ **Corollary 24.** *Let $G = (V, E)$ with $s, t \in V$ be an n-vertex complete graph and suppose we are promised that $G(x)$ has a path of length L for $L \in [3, n/4]$ between s and t. Then st-PATH$_G(x)$ has randomized query complexity $\Omega(n^2)$.*

4.3.1 Graph with a Single Path

When the graph $G(x)$ is known to have a single st-path, we will we use a divide-and-conquer algorithm to find the path. To show that the divide-and-conquer approach is useful, we first consider the simpler algorithm (as described above) that uses Theorem 19 to find an edge $\{u, v\}$ on the path, and then once that edge is found, the algorithm is run on a new graph where vertices u and v are identified. This process is continued until the edge $\{s, t\}$ is found. Thus if the length of the path is initially L, after an edge is found, the path length will be $L-1$, and then $L-2$ in the next iteration, etc. Ignoring error, and assuming the algorithm finds an edge in each round, by Theorem 19, the query complexity at the ith round will be $\widetilde{O}(n\sqrt{L-i})$. Over the course of the L rounds, the total query complexity will be

$$\sum_{i=0}^{L-1} \widetilde{O}(n\sqrt{L-i}) = \widetilde{O}\left(nL^{3/2}\right). \tag{51}$$

For $L \geq n^{2/3}$, this algorithm does not even outperform the best classical algorithm, and for $L \geq n^{1/3}$ it does not outperform the quantum algorithm of Ref. [18].

We instead consider the following divide-and-conquer approach, described in detail in Algorithm 3. We use Algorithm 2 to find a set of edges, some of which are very likely to be on the path. Then we use Lemma 8 to verify which of those edges is actually on the path, and Lemma 9 to ensure we choose an edge near the center of the path, so we are left with two subproblems of approximately half the size. Finally two recursive calls find the unique path from s to the found edge, and the unique path from t to the found edge.

▶ **Theorem 25.** *Let $p \geq 0$, and $G = (V, E)$ with $s, t \in V$ be a family of n-vertex graphs, and suppose we are promised that $G(x)$ contains a single st-path of some length L (L may depend on x and need not be known ahead of time). Then there is a quantum algorithm (Algorithm 3) that with probability $1 - O(p)$ solves st-PATH$_G(x)$ and uses $\widetilde{O}(nL^{1+o(1)} \log^2(1/p))$ expected queries on input x.*

Proof. We first analyze the probability of error, then we prove the correctness of Algorithm 3, assuming that no errors are made, and finally, we analyze the query complexity.

We will stop the algorithm after $O(n)$ recursive calls. Since each recursive call returns an edge, and any path has length at most n, this termination will not affect the success probability. We then bound our probability of error by $O(p/n^4) = O(p)$, by showing that the failure probability in each recursive call is $O(p/n^5)$.

We say a failure occurs (in some recursive call) if any of the following happens:

1. Any one of the at most 4ℓ `PathDetection` algorithms errs. This has probability $O(\ell\delta) = O(p/n^5)$, by our choice of $\delta = p/(\ell n^5)$.
2. One of the at most $O(\ell)$ calls to `WitnessSizeEst` produces an estimate that is not within the desired relative error. This has probability $O(\ell\delta) = O(p/n^5)$.
3. None of the ℓ iterations of `EdgeFinder` produces an edge that is on the st-path, and moreover, that is within $(\varepsilon_3 - \varepsilon_2)L = \sqrt{\varepsilon_1}L$ of the middle of the path. The absence of this type of failure is sufficient to guarantee that the condition on Line 20 will be satisfied, as long as `WitnessSizeEst` is also successful.

We analyze the probability of the last event, assuming the first two do not occur. Let $e_0, \ldots, e_{L-1}$ denote the path edges, in order, in the unique st-path in $G(x)$. For one of the ℓ runs of `EdgeFinder`, the probability that it does not output "Failure" is ε_1. Conditioned on

Algorithm 3 `SinglePathFinder`(O_x, p, G, s, t).

Input : Failure tolerance $p > 0$, oracle O_x for the graph $G(x) = (V, E(x))$, $s, t \in V$ such that there is a unique path from s to t.

Output : With probability $1 - O(p)$, a set of edges whose vertices form a path from s to t in $G(x)$.

`// Base Cases`

if $s = t$ **then** Return $\emptyset$

if $\{s, t\} \in E(x)$ **then** Return $\{s, t\}$

`// Finding Possible Edges on Path`

$\varepsilon_1 \leftarrow \frac{1}{\log n}$ `// Any` $\varepsilon_1 = o(1)$ `that is inverse polylog`(n) `would suffice`

$S \leftarrow \emptyset$

$\ell \leftarrow \frac{2\log(n^5/p)}{\varepsilon_1}$ **for** $i = 1$ **to** ℓ **do**

  $e \leftarrow$ `EdgeFinder`$(O_x, \varepsilon_1, G, s, t)$ (Algorithm 2)

  if $e \neq$ *"Failure" and* $e = (u, v) \in \overrightarrow{E}(x)$ **then** $S = S \cup \{(u, v), (v, u)\}$

`// Finding a possible edge that is actually on a path`

$\delta \leftarrow p/(\ell n^5)$

for $(u, v) \in S$ **do**

  Initialize `PathDetection`$(O_x, G^-_{\{u,v\}}, s, u, \delta)$ (Lemma 8)

  Initialize `PathDetection`$(O_x, G^-_{\{u,v\}}, v, t, \delta)$

$flag \leftarrow$ True

while *flag* **do**

  Run in parallel each `PathDetection` algorithm initialized in the prior **for** loop, until each algorithm applies O_x once or terminates (or do nothing for those algorithms that have terminated previously)

  for $(u, v) \in S$ **do**

    if `PathDetection`$(O_x, G^-_{\{u,v\}}, s, u, \delta)$ *and* `PathDetection`$(O_x, G^-_{\{u,v\}}, v, t, \delta)$ *have both terminated in this iteration of the* ***while*** *loop and both detected paths* **then**

      $\varepsilon_2 \leftarrow \sqrt{\varepsilon_1}$, $\varepsilon_3 \leftarrow 2\sqrt{\varepsilon_1}$

      $\tilde{k} \leftarrow$`WitnessSizeEst`$(O_x, G, s, u, \varepsilon_2, \delta)$ (Lemma 9) `// estimate of dist.` s `to` u

      if $|\tilde{k} - L/2| \leq \varepsilon_3 L$ **then**

        $(u^*, v^*) \leftarrow (u, v)$

        $flag \leftarrow$ False

`// Recursive call`

Return $\{(u^*, v^*)\} \cup$ `SinglePathFinder`$(O_x, p, G, s, u^*) \cup$ `SinglePathFinder`(O_x, p, G, v^*, t)

the output of `EdgeFinder` not being "Failure," by Theorem 19, we sample from a distribution $\hat{q}$ that is $\sqrt{\varepsilon_1}$-close in total variation distance to the uniform distribution over edges on the st-path. Thus, the probability that we sample an edge in the set

$$R = \{e_k : k \in [L/2 - (\varepsilon_3 - \varepsilon_2)L, L/2 + (\varepsilon_3 - \varepsilon_2)L]\}, \tag{52}$$

where e_k is the k^{th} path edge, is:

$$\hat{q}(R) \geq \frac{|R|}{L} - \sqrt{\varepsilon_1} = 2(\varepsilon_3 - \varepsilon_2) - \sqrt{\varepsilon_1} = 2(2\sqrt{\varepsilon_1} - \sqrt{\varepsilon_1}) - \sqrt{\varepsilon_1} = \sqrt{\varepsilon_1}. \tag{53}$$

Thus, using $\varepsilon_1 \leq 1/2$, each of the ℓ samples has probability at least $(1 - \varepsilon_1)\sqrt{\varepsilon_1} \geq \sqrt{\varepsilon_1}/2$ of being a path edge in the correct range, R. Using Hoeffding's bound, the probability that none of them is a path edge in the correct range is thus at most:

$$e^{-2\ell(\sqrt{\varepsilon_1}/2)^2} = e^{-\ell\varepsilon_1/2} = e^{-\log(n^5/p)} = O(n^{-5}p) \tag{54}$$

by our choice of $\ell = 2\log(n^5/p)/\varepsilon_1$. The total probability of failure in one round is thus at most $O(p/n^5)$.

We prove correctness using induction on L, the length of the path, assuming no failure occurs. For the base case, if $L \in \{0, 1\}$, we will correctly return the path in Lines 1 and 2.

For the inductive case, let $L' \geq 1$. We assume `SinglePathFinder` works correctly for all lengths L such that $0 \leq L \leq L'$. Now consider a graph with $L = L' + 1$. Then assuming no failure, we will sample at least one edge (u, v) in the set $R = \{e_k : k \in [L/2 - (\varepsilon_3 - \varepsilon_2)L, L/2 + (\varepsilon_3 - \varepsilon_2)L]\}$ (not doing so is a failure of the type specified by Item 3 in the list above). Then if there are no errors in the `PathDetection` algorithms, Line 16 will be satisfied when (u, v) corresponds to an edge in the path where u is closer to s and v is closer to t. This is because we have removed $\{u, v\}$ from the graph when we are running `PathDetection`, and since there is a unique st-path, there will only be a path from s to u and not from s to v, and likewise for t.

Then for every edge (u, v) that we have correctly found using `PathDetection` to be on a path, we apply `WitnessSizeEst` (see Lemma 9) to estimate $R_{s,u}(G(x))$. If $(u, v) = e_k$, then $e_0, \ldots, e_{k-1}$ is the unique su-path in G, and it has length k, and so $R_{s,u}(G(x)) = k$, and thus `WitnessSizeEst` is actually estimating k. Assuming $(u, v) \in R$, (and we know this holds for at least one such edge), we have $|k - L/2| \leq (\varepsilon_3 - \varepsilon_2)L$. Then since we assume `WitnessSizeEst` does not fail, it outputs an estimate $\tilde{k}$ of k, such that $|\tilde{k} - k| \leq \varepsilon_2 k \leq \varepsilon_2 L$. Together, these conditioned imply $|\tilde{k} - L/2| \leq \varepsilon_3 L$, which will trigger the **while** loop to halt. It is possible that we will break out of the loop for an edge not in R, but at the least we know that if no failure occurs, we will will certainty break out of the **while** loop with an edge (u^*, v^*) on the path.

Now that we have the edge (u^*, v^*), to find the rest of the path, we just need to find the rest of the path from s to u^* and from v^* to t. But both of these problems will have path lengths between 0 and L', so by inductive assumption, the recursive calls in Line 23 will be correct, and will return the edges on the paths.

Turning to our analysis of the expected query complexity, we first bound the contribution to the expected query complexity in the case of a failure. As just discussed, a failure occurs with probability $O(p/n^4)$. Even in case of failure, each of our $O(n\log(n/p)) = O(n^2\log(1/p))$ calls to `EdgeFinder`, `PathDetection`, and `WitnessSizeEst` still has expected query complexity at most $\widetilde{O}(n^{1.5}(1/\varepsilon_1 + 1/\varepsilon_2^{3/2})\log(1/\delta)) = O(n^2\log(1/p))$ (for *any* x), for a total query cost of $O(n^4\log^2(1/p))$. Thus, the error case contributes an additive $O(p\log^2(1/p)) = O(1)$ to the expected query complexity.

Next, we create a recurrence relation for the expected query complexity, assuming no failure occurs. Let $\mathbb{E}[T_L]$ be the expected query complexity of Algorithm 3 on a graph with n vertices, when there is a single path, and that path has length L. For $k \in \{0, \ldots, L-1\}$, let $\tilde{q}_L(k)$ be the probability that the path edge that we find, (u^*, v^*), is e_k. Because we assume no subroutine call fails, we can assume that $\tilde{k}$ is an estimate of k with relative error ε_2, so $|\tilde{k}-k| \leq \varepsilon_2 k \leq \varepsilon_2 L$. From the conditional statement in Line 20, we also have $|\tilde{k}-L/2| \leq \varepsilon_3 L$. Taken together, these imply:

$$|k - L/2| \leq (\varepsilon_2 + \varepsilon_3)L = (\sqrt{\varepsilon_1} + 2\sqrt{\varepsilon_1})L = 3\sqrt{\varepsilon_1}L. \tag{55}$$

Thus with certainty (assuming no failure occurs), we will exit the **while** loop with $(u^*, v^*) = e_k$, for $k \in [(1/2 - 3\sqrt{\varepsilon_1})L, (1/2 + 3\sqrt{\varepsilon_1})L]$, so:

$$\mathbb{E}[T_L] = \widetilde{O}(\ell n\sqrt{L}/\varepsilon_1) + \widetilde{O}(\ell n\sqrt{L}\log(1/\delta)) + \widetilde{O}\left(\ell \frac{n\sqrt{L}}{\varepsilon_2^{3/2}}\log(1/\delta)\right) + \sum_{k=\lceil(1/2-3\sqrt{\varepsilon_1})L\rceil}^{\lfloor(1/2+3\sqrt{\varepsilon_1})L\rfloor} \tilde{q}_L(k)\left(\mathbb{E}[T_k] + \mathbb{E}[T_{L-k-1}]\right), \tag{56}$$

where the first three terms come from: (1) running `EdgeFinder` (Algorithm 2, Theorem 19) ℓ times; (2) at most $O(\ell)$ parallel `PathDetection` (Lemma 8) algorithms; and (3) running `WitnessSizeEst` (Lemma 9) $O(\ell)$ times; and the final term from the two recursive calls.

To get a function that is strictly increasing in L, let $T'_L := \max_{k \leq L} \mathbb{E}[T_k]$, so in particular $\mathbb{E}[T_L] \leq T'_L$, and T'_L also satisfies the recursion in Equation (56) (with $=$ replaced by $\leq$). Then we have, for any $k \in [(1/2 - 3\sqrt{\varepsilon_1})L, (1/2 + 3\sqrt{\varepsilon_1})L]$,

$$\mathbb{E}[T_k] + \mathbb{E}[T_{L-k-1}] \leq 2T'_{(1/2+3\sqrt{\varepsilon_1})L}. \tag{57}$$

Continuing from Equation (56), and using $1/\varepsilon_1 = \log n$ and $1/\varepsilon_2 = 1/\sqrt{\varepsilon_1} = \sqrt{\log n}$, $\ell = 2\log(n^5/p)/\varepsilon_1 = O(\log(1/p)\log^2 n)$, and $\log(1/\delta) = O(\log(\ell n/p)) = \log(1/p)\mathrm{polylog}(n, \log(1/p))$, we get

$$\mathbb{E}[T_L] \leq T'_L \leq \widetilde{O}\left(n\sqrt{L}\log^2(1/p)\right) + 2T_{(1/2+3\sqrt{\varepsilon_1})L}. \tag{58}$$

To analyze this recurrence, we add up the queries made in every recursive call. At the i^{th} level of recursion, there are 2^i recursive calls, and each one makes $\widetilde{O}\left(n\sqrt{L/b^i}\log^2(1/p)\right)$ queries itself, where $b = (1/2 + 3\sqrt{\varepsilon_1})^{-1}$, before recursing further. Thus

$$\begin{aligned}\mathbb{E}[T_L] &\leq \widetilde{O}\left(n\sqrt{L}\log^2(1/p)\right) + \sum_{i=1}^{\log_b L} 2^i\sqrt{\frac{L}{b^i}} \cdot \widetilde{O}\left(n\log^2(1/p)\right) \\ &\leq \widetilde{O}\left(n\sqrt{L}\log^2(1/p)\right) + \widetilde{O}\left(n\sqrt{L}\log^2(1/p)\right)\left(2/\sqrt{b}\right)^{\log_b L}.\end{aligned} \tag{59}$$

Letting $\eta := \frac{1}{1+\frac{1}{6\sqrt{\varepsilon_1}}} = O(1/\sqrt{\log n})$ since $\varepsilon_1 = 1/\log n$, so that $b = 2(1-\eta)$, we have:

$$\begin{aligned}\log\left(2/\sqrt{b}\right)^{\log_b L} &= \left(1 - \frac{1}{2}\log b\right)\frac{\log L}{\log b} = \left(\frac{1}{\log b} - \frac{1}{2}\right)\log L \\ &= \left(\frac{1}{1 - \log\frac{1}{1-\eta}} - \frac{1}{2}\right)\log L = \left(\frac{1}{2} + \frac{\log\frac{1}{1-\eta}}{1 - \log\frac{1}{1-\eta}}\right)\log L \\ \text{so } \left(2/\sqrt{b}\right)^{\log_b L} &= L^{\frac{1}{2}+o(1)},\end{aligned} \tag{60}$$

where we used $\frac{\log \frac{1}{1-\eta}}{1-\log \frac{1}{1-\eta}} = o(1)$, since $\log \frac{1}{1-\eta} = o(1)$, which follows from $\eta = o(1)$. Thus, continuing from Equation (59), we have:

$$\mathbb{E}[T_L] = \widetilde{O}\left(n\sqrt{L}\log^2(1/p)\right) L^{\frac{1}{2}+o(1)} = \widetilde{O}\left(nL^{1+o(1)}\log^2(1/p)\right). \tag{61}$$

We note that while our approach in Theorem 25 outperforms the simpler, non-divide-and-conquer algorithm analyzed in Equation (51), it performs worse than the algorithm of Ref. [18] for graphs with $L = \Omega(n^{1/2-o(1)})$. Thus, one could run Algorithm 3 until $O\left(n^{3/2}\right)$ queries had been made, and then switch to the algorithm of Ref. [18].

4.3.2 Path Finding in Arbitrary Graphs

When $G(x)$ is not known to only have one st-path, while it is possible that an algorithm similar to Algorithm 3 would solve st-$\text{PATH}_G(x)$, we have not been able to bound the running time effectively. This is because in the case of a single path, once you find an intermediate edge on the path, the longest paths from s and t to that edge must be shorter than the length of the longest path from s to t. This ensures that subproblems take shorter time than the original problem. With multiple paths, we no longer have that guarantee.

However, we provide an alternative approach that, while not as fast as Algorithm 3, still provides an improvement over the algorithm of [18] for graphs in which all (self-avoiding) paths from s to t are short. Our approach does not make use of our path-edge sampling algorithm as a subroutine, and instead uses the path detection algorithm of Lemma 8 to decide whether there are paths through various subgraphs, and then uses that information to find each edge in a path in order from s to t. In this way, we avoid the problem of subproblems being larger than the original problem, since if the longest path from s to t has length L, and the first edge we find on the path is (s, u), then longest path from u to t that doesn't go through s must have length at most $L - 1$. However, we lose the advantage of a divide-and-conquer approach.

To find the first edge on a path, we use a group testing approach. We divide the neighbors of s in G into two sets, S_1 and S_2 and run path detection algorithms in parallel on two subgraphs of $G(x)$, one with edges from s removed, except those to vertices in S_1 (that is, $G^-_{\{\{s,u\}\in E : u\in S_1\}}$), and one with edges from s removed, except those to vertices in S_2. We will detect which of these subgraphs contains a path, and we will know there is a path whose first edge goes from s to a vertex in the corresponding set (S_1 or S_2). Then we divide that set into half again, and repeat, until we have narrowed down our set to one vertex u, that must be the first vertex on a path from s to t.

At this point we have learned the first edge on a path from s to t. We then consider G^-_s, which is G with vertex s removed, and recursively iterate this procedure to learn the first edge on a path from u to t. Using this approach, we obtain the following result:

▶ **Theorem 26.** *Let $p \geq 0$, and $G = (V, E)$ with $s, t \in V$ be a family of n-vertex graphs, and suppose we are promised that there is a path from s to t in $G(x)$. On input x, if the longest st-path in $G(x)$ has length L (L need not be known ahead of time), there is a quantum algorithm that returns the edges on a path with probability $1-O(p)$ and uses $\widetilde{O}(nL^{3/2}\log(1/p))$ expected queries.*

A detailed description of the algorithm and the proof of Theorem 26 can be found in the full version of this work [26, Section 4.3.2].

References

1 Scott Aaronson. Open problems related to quantum query complexity. *ACM Transactions on Quantum Computing*, 2(4):1–9, 2021.

2 Noel T. Anderson, Jay-U Chung, Shelby Kimmel, Da-Yeon Koh, and Xiaohan Ye. Improved quantum query complexity on easier inputs. *arXiv preprint arXiv:2303.00217*, 2023.

3 Simon Apers and Stephen Piddock. Elfs, trees and quantum walks. *arXiv preprint arXiv:2211.16379*, 2022.

4 Salman Beigi and Leila Taghavi. Quantum speedup based on classical decision trees. *Quantum*, 4:241, 2020.

5 Salman Beigi, Leila Taghavi, and Artin Tajdini. Time- and query-optimal quantum algorithms based on decision trees. *ACM Transactions on Quantum Computing*, 3(4):1–31, 2022.

6 Aleksandrs Belovs. Learning-graph-based quantum algorithm for k-distinctness. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS 2012)*, pages 207–216. IEEE, 2012.

7 Aleksandrs Belovs and Ben W. Reichardt. Span programs and quantum algorithms for st-connectivity and claw detection. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA 2012)*, pages 193–204. Springer, 2012.

8 Chris Cade, Ashley Montanaro, and Aleksandrs Belovs. Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness. *Quantum Information & Computation*, 18(1-2):18–50, 2018.

9 Titouan Carette, Mathieu Laurière, and Frédéric Magniez. Extended learning graphs for triangle finding. *Algorithmica*, 82(4):980–1005, 2020.

10 Ashok K. Chandra, Prabhakar Raghavan, Walter L. Ruzzo, Roman Smolensky, and Prasoon Tiwari. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity*, 6(4):312–340, 1996.

11 Denis X. Charles, Kristen E. Lauter, and Eyal Z. Goren. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22:93–113, 2007. doi:10.1007/s00145-007-9002-x.

12 Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003)*, pages 59–68, 2003.

13 Andrew M. Childs, Matthew Coudron, and Amin Shiraz Gilani. Quantum algorithms and the power of forgetting. *arXiv preprint arXiv:2211.12447*, 2022.

14 Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, 1998. doi:10.1098/rspa.1998.0164.

15 Arjan Cornelissen, Stacey Jeffery, Maris Ozols, and Alvaro Piedrafita. Span programs and quantum time complexity. In *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, pages 26:1–26:14, 2020.

16 Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematial Cryptology*, 8:209–247, 2014. doi:10.1515/jmc-2012-0015.

17 Kai DeLorenzo, Shelby Kimmel, and R. Teal Witter. Applications of the Quantum Algorithm for st-Connectivity. In *Proceedings of the 14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, volume 135 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:14, 2019. doi:10.4230/LIPIcs.TQC.2019.6.

18 Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006.

19 Steven D. Galbraith and Frederik Vercauteren. Computational problems in supersingular elliptic curve isogenies. *Quantum Information Processing*, 17(265), 2018. doi:10.1007/s11128-018-2023-6.

20 Dmitry Grinko, Julien Gacon, Christa Zoufal, and Stefan Woerner. Iterative quantum amplitude estimation. *npj Quantum Information*, 7(1):52, March 2021. doi:10.1038/s41534-021-00379-1.

21 Tsuyoshi Ito and Stacey Jeffery. Approximate Span Programs. *Algorithmica*, 81(6):2158–2195, 2019. doi:10.1007/s00453-018-0527-1.

22 Michael Jarret, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. Quantum Algorithms for Connectivity and Related Problems. In *Proceedings of the 26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:13, 2018. doi:10.4230/LIPIcs.ESA.2018.49.

23 Stacey Jeffery. Quantum subroutine composition. *arXiv preprint arXiv:2209.14146*, 2022.

24 Stacey Jeffery. Span programs and quantum space complexity. *Theory of Computing*, 18(1):1–49, 2022.

25 Stacey Jeffery and Shelby Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1:26, 2017. doi:10.22331/q-2017-08-17-26.

26 Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. Quantum algorithm for path-edge sampling. *arXiv preprint arXiv:2303.03319*, 2023.

27 Stacey Jeffery and Sebastian Zur. Multidimensional quantum walks, with application to k-distinctness. *arXiv preprint arXiv:2208.13492*, 2022.

28 Mauricio Karchmer and Avi Wigderson. On span programs. In *Proceedings of the 8th Annual IEEE Conference on Structure in Complexity Theory*, pages 102–111, 1993.

29 A. Yu Kitaev. Quantum measurements and the Abelian Stabilizer Problem. *arXiv:quant-ph/9511026*, 1995. arXiv:quant-ph/9511026.

30 Troy Lee, Frédéric Magniez, and Miklos Santha. A learning graph based quantum query algorithm for finding constant-size subgraphs. *Chicago Journal of Theoretical Computer Science*, 18(1), 2011. doi:10.4086/cjtcs.2012.010.

31 Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Špalek, and Mario Szegedy. Quantum Query Complexity of State Conversion. In *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2011)*, pages 344–353, 2011. doi:10.1109/FOCS.2011.75.

32 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via Quantum Walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. doi:10.1137/090745854.

33 Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

34 Ben W. Reichardt. Reflections for quantum query algorithms. In *Proceedings of the 2011 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 560–569, 2011. doi:10.1137/1.9781611973082.44.

35 Ben W. Reichardt. Span programs are equivalent to quantum query algorithms. *SIAM Journal on Computing*, 43(3):1206–1219, 2014. doi:10.1137/100792640.

36 Seiichiro Tani. Claw finding algorithms using quantum walk. *Theoretical Computer Science*, 410:5285–5297, 2009. doi:10.1016/j.tcs.2009.08.030.