Applications of the Quantum Algorithm for st-Connectivity

Kai DeLorenzo
Middlebury College, Computer Science Department, Middlebury, VT, USA

Shelby Kimmel
Middlebury College, Computer Science Department, Middlebury, VT, USA

R. Teal Witter
Middlebury College, Computer Science Department, Middlebury, VT, USA

Abstract

We present quantum algorithms for various problems related to graph connectivity. We give simple and query-optimal algorithms for cycle detection and odd-length cycle detection (bipartiteness) using a reduction to st-connectivity. Furthermore, we show that our algorithm for cycle detection has improved performance under the promise of large circuit rank or a small number of edges. We also provide algorithms for detecting even-length cycles and for estimating the circuit rank of a graph. All of our algorithms have logarithmic space complexity.

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

Keywords and phrases graphs, algorithms, query complexity, quantum algorithms, span programs

Digital Object Identifier 10.4230/LIPIcs.TQC.2019.6

Funding This research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-18-1-0286. 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.

Acknowledgements We thank Chris Cade and Stacey Jeffery for helpful discussions.

1 Introduction

Quantum query algorithms are remarkably described by span programs [15, 16], a linear algebraic object originally created to study classical logspace complexity [12]. However, finding optimal span program algorithms can be challenging; while they can be obtained using a semidefinite program, the size of the program grows exponentially with the size of the input to the algorithm. Moreover, span programs are designed to characterize the query complexity of an algorithm, while in practice we also care about the time and space complexity.

One of the nicest span programs is for the problem of undirected *st*-connectivity, in which one must decide whether two vertices s and t are connected in a given graph. It is "nice" for several reasons:

- It is easy to describe and understand why it is correct.
- It corresponds to a quantum algorithm that uses logarithmic (in the number of vertices and edges of the graph) space [4, 11].
- The time complexity of the corresponding algorithm is the product of the query complexity, and the time required to implement a unitary that applies one step of a quantum walk on the underlying graph [4, 11]. On the complete graph, for example, the quantum walk step introduces only an additional logarithmic factor to the complexity [4].

- The query complexity of the algorithm is determined by two well known graph functions, the effective resistance and effective capacitance [10].

Thus one strategy for designing other "nice" quantum algorithms is to reduce a given problem to *st*-connectivity, and then use the span program *st*-connectivity algorithm. This strategy has proven to be quite successful, and in fact has produced several optimal or nearly optimal algorithms. There is a reduction from Boolean formula evaluation to *st*-connectivity [14] that produces an optimal quantum algorithm for read-once formulas [11]. There is an optimal reduction from graph connectivity to *st*-connectivity [10]. Cade et al. use an *st*-connectivity subroutine to create nearly query-optimal algorithms for cycle detection and bipartiteness [7][1]. Finally, the *st*-connectivity span program algorithm underlies the learning graph framework [3], one of the most successful heuristics for span program algorithm design.

In this work, we follow precisely this strategy for creating "nice" quantum algorithms: we reduce the graph problems of cycle detection, odd-length path detection, bipartiteness, and even-length cycle detection to *st*-connectivity. In our reductions, solving the related *st*-connectivity problem comprises the whole algorithm; in other words, we create a new graph that has an *st*-path if and only if the original graph has the property in question.

Additionally, there is an an estimation algorithm closely related to the *st*-connectivity algorithm that determines the size of the effective resistance or effective capacitance of the graph [9, 10]. Not only is it often useful to estimate the effective resistance or effective capacitance of a graph, as these quantities bound the shortest path length and smallest cut size respectively, but sometimes one can encode quantities of interest as either the effective capacitance or effective resistance. For example, given a graph G, it is possible to create a new graph whose effective resistance is the average effective resistance (Kirchoff index) of the original graph [10]. Using this strategy of reduction to effective resistance estimation, we also create an algorithm to estimate the circuit rank of a graph.

1.1 Contributions and Comparison to Previous Work

All of our algorithms are in the adjacency matrix model (see Section 2), in which one can query elements of the adjacency matrix of the input graph. This contrasts with work such as [7] which study similar problems in the adjacency list model.

We note all of our algorithms are space efficient, in that the number of qubits required are logarithmic in the number of edges and vertices of the graph. (This property is inherited directly from the basic *st*-connectivity span program.) We do not analyze time complexity, but, as mentioned above, it is the product of the query complexity, which we analyze, and the time required to implement certain quantum walk unitaries. (We leave this analysis for future work.)

We next discuss the context of each of our results in turn. In this section, we assume the underlying graph is the complete graph on n vertices to more easily compare to previous work, although in the main body of the paper, we show our results apply more to generic underlying graphs with n vertices and m edges.

Cycle Detection

Cade et al. [7] describe a nearly optimal $\tilde{O}(n^{3/2})$ quantum query algorithm for cycle detection via reduction to *st*-connectivity, almost matching the lower bound of $\Omega(n^{3/2})$ [8]. In this work, we find an algorithm that removes the log-factors of the previous result, giving an

[1] In Ref. [2], Āriņš designed algorithms for connectivity and bipartiteness which, while not strictly reductions to *st*-connectivity, are very closely related.

algorithm with optimal $O(n^{3/2})$ query complexity. Moreover, our algorithm is simpler than that in Ref. [7]: their approach requires solving an *st*-connectivity problem within a Grover search, while our approach is entirely based on solving an *st*-connectivity problem.

We furthermore prove that if promised that (in the case of a cycle) the circuit rank of the graph is at least r or (in the case of no cycles) there are at most μ edges, then the query complexity of cycle detection is $O(\mu\sqrt{n/r})$.

Bipartiteness

An optimal quantum query algorithm for bipartiteness was created by Ārinš [2], matching the lower bound of $\Omega(n^{3/2})$ [19]. However, this algorithm was not known to be time efficient (and did not use a reduction to *st*-connectivity, although the ideas are quite similar to the approach here and in [7]). In Ref. [7], Cade et al., using similar ideas as in their cycle detection algorithm, create a bipartiteness checking algorithm using a reduction to *st*-connectivity that is again embedded in a search loop, which is optimal up to logarithmic factors in query complexity. Our algorithm for bipartiteness removes the logarithmic factors of [7], and so recovers the optimal query complexity of [2], while retaining the simplicity of the reduction to *st*-connectivity.

Even-length Cycle Detection

The problem of detecting an even-length cycle can provide insight into the structure of the graph. For example, it is straightforward to see that in a graph with no even cycles, no edge can be involved in more than one cycle. Classically this problem requires $\Theta(n^2)$ queries [17]. We are not aware of an existing quantum algorithm for this problem; we provide an $O(n^{3/2})$ query algorithm.

Estimation of Circuit Rank

Circuit rank parameterizes the number of cycles in a graph: it is the number of edges that must be removed before there are no cycles left in a graph. It has also been used to describe the complexity of computer programs [13]. We give an algorithm to estimate the circuit rank r to multiplicative error ϵ with query complexity $\widetilde{O}\left(\epsilon^{-3/2}\sqrt{n^4/r}\right)$ in the generic case and query complexity $\widetilde{O}\left(\epsilon^{-3/2}\sqrt{n^3/r}\right)$ when promised that the graph is a cactus graph. When additionally promised that the circuit rank is large, these algorithms can have non-trivial query complexity. We are aware of no other classical or quantum query algorithms that determine or estimate this quantity.

Odd-length Path

We provide an algorithm to determine whether there is an odd-length path between two specified vertices that uses $O(n^{3/2})$ queries. While perhaps not the most interesting problem on its own, we effectively leverage this algorithm as a subroutine in several of our other constructions.

1.2 Open Problems

Throughout this paper, our strategy is to take a graph G, use it to create a new graph G', such that there is an *st*-path through G' if and only if G has a certain property. We analyze the query complexity of the algorithms we create in detail, but not the time complexity. The

time complexities of our algorithms depend on the time required to implement one step of a quantum walk on the graph G' (see U from Theorem 4 and Ref. [11]) [2]. We strongly suspect that the highly structured nature of the graphs G' we consider would yield time-efficient algorithms, but we have not done a full analysis.

In the case of our algorithm for cycle detection, we create a graph G' whose effective resistance is the circuit rank of the original graph G. For the graphs we design for the other algorithms in this paper, do the effective resistance or effective capacitance have relevant meanings?

The query complexity of our algorithm for estimating the circuit rank of a graph depends on bounding a quantity called the approximate negative witness. While the best bound we currently have is $O(n^4)$, we believe this is not tight. Obtaining a better bound would not only be interesting for these results, but could provide insight into more general quantum estimation algorithms.

Several of our results rely on a graph that tests for paths of odd length, i.e. those whose length modulo 2 is 1. Is there a way to adapt our algorithm to test for paths of arbitrary modulus?

Ref. [1] provides a list of complete problems for symmetric logarithmic space (SL), of which *st*-connectivity is one such problem. It would be interesting to study the query complexity of these problems to see if reducing to *st*-connectivity always gives an optimal approach.

Finally, it would be nice to improve the quantum lower bounds and classical bounds for several of these problems. In particular, we would like to obtain better quantum lower bounds for the even-length cycle detection and odd-length path detection problems. (For the later, the best quantum lower bound is $\Omega(n)$ [4].) We expect that the promise of large circuit rank also aids a classical algorithm for cycle detection, and it would be interesting to know by how much, in order to compare to our quantum algorithm.

2 Preliminaries

We consider undirected graphs $G = (V, E)$ where V is a set of vertices and E a set of edges; we often use $E(G)$ and $V(G)$ to denote the sets of edges and vertices of a graph G when there are multiple graphs involved. If clear which graph we are referring to, we will use n for the number of vertices in the graph, and m for the number of edges. For ease of notation, we associate each edge with a unique label ℓ. For example, we refer to an edge between vertices u and v labeled by ℓ as $\{u, v\}$, or simply as ℓ. In general, we could consider a weighting function on the edges or consider graphs with multi-edges (see [10] for more details on how these modifications are implemented) but for our purposes, we will always consider all edges to have weight 1, and we will only consider graphs with a at most a single edge between any two vertices.

We will use the following notation regarding spanning trees: if G is a connected graph and $\ell \in E(G)$, then we use $t_\ell(G)$ to denote the number of unique spanning trees of G that include edge ℓ, and we use $t(G)$ to denote the total number of unique spanning trees of G.

In Section 3 we describe a quantum algorithm for estimating the circuit rank of a graph, which is a quantity that is relevant for a number of applications, like determining the robustness of a network, analyzing chemical structure [18], or parameterizing the complexity of a program [13].

[2] In Ref. [7], they claim that their algorithms are time efficient, but their time analysis only considers the case of the underlying graph being the complete graph, while the actual graph used in their algorithms is not the complete graph. We expect that their algorithms can be implemented efficiently, but more work is needed to show this.

▶ **Definition 1** (Circuit Rank). *The* circuit rank *of a graph with m edges and n vertices is $m - n + \kappa$ where κ is the number of connected components. Alternatively, the circuit rank of a graph is the minimum number of edges that must be removed to break all cycles and turn the graph into a tree or forest.*

We also consider a special class of graphs called cactus graphs:

▶ **Definition 2.** *A cactus graph is a connected graph in which any two simple cycles share at most one common vertex.*

We will in particular use cactus forests, in which all components of the graph are cacti. For cactus forests, the circuit rank is simply the total number of cycles in the graph.

The final type of graph we need is the bipartite double graph:

▶ **Definition 3.** *Given a graph G, the bipartite double of G, denoted K^G, is the graph that consists of two copies of the vertices of G (with vertex $v \in V(G)$ labeled as v_0 in the first copy and v_1 in the second), with all original edges removed, and edges $\{u_0, v_1\}$ and $\{v_0, u_1\}$ created for each edge $\{u, v\} \in E(G)$. This graph is also known as the Kronecker cover of G, or $G \times K_2$, and its adjacency matrix is given by $G \otimes X$ (where X is the Pauli X operator.)*

We associate edges of G with literals of a string $x \in \{0,1\}^N$, where $N \leq |E|$, and a literal is either x_i or $\overline{x}_i$ for $i \in [N]$. The subgraph $G(x)$ of G contains an edge ℓ if ℓ is associated with x_i and $x_i = 1$, or if ℓ is associated with $\overline{x}_i$ and $x_i = 0$. (See [10] for more details on this association.)

We assume that we have complete knowledge of G, and access to $x \in \{0,1\}^N$ via a black box unitary (oracle) O_x. This oracle acts as $O_x|i\rangle|b\rangle = |i\rangle|b \oplus x_i\rangle$, where x_i is the i^{th} bit of x. Then our goal is to use O_x as few times as possible to determine a property of the graph $G(x)$. The number of uses of O_x required for a given application is called the query complexity.

We will be applying and analyzing an algorithm for st-connectivity, which is the problem of deciding whether two nodes $s, t \in V(G)$ are connected in a graph $G(x)$, where G is initially known, but x must be determined using the oracle, and we are promised $x \in X$, where $X \subseteq \{0,1\}^N$. An st-path is a series of edges connecting s to t.

Given two instances of st-connectivity, they can be combined in *parallel*, where the two s vertices are identified and labeled as the new s, and the two t vertices are identified and labeled as the new t. This new st-connectivity problem encodes the logical OR of the original connectivity problems, in that the new graph is connected if and only if at least one of the original graphs was connected. Two instances of st-connectivity can also be combined in *series*, where the s vertex of one graph is identified with the t vertex of the other graph, and relabeled using a label not previously used for a vertex in either graph. This new st-connectivity encodes the logical AND of the original connectivity problems, in that the new graph is connected if and only if both of the original graphs were connected. This correspondence was noted in [14] and applied to quantum query algorithms for Boolean formulas in [11] and for determining total connectivity in [10].

The key figures of merit for determining the query complexity of the span-program-based st-connectivity quantum algorithm are effective resistance and effective capacitance [10]. We use $R_{s,t}(G(x))$ to denote the effective resistance between vertices s and t in a graph $G(x)$, and we use $C_{s,t}(G(x))$ to denote the effective capacitance between vertices s and t in $G(x)$. These two functions were originally formulated to characterize electrical circuits, but are also important functions in graph theory. For example, effective resistance is related to the hitting time of a random walk on a graph. (See [6] for more information on effective resistance and capacitance in the context of graph theory.) For this paper, we don't require a formal definition of these functions, but can instead use a few of their well known and easily derived properties (see [10] for formal definitions):

Effective Resistance

1. If G consists of a single edge between s and t, and $G(x) = G$, then $R_{s,t}(G(x)) = 1$.
2. If s and t are not connected in $G(x)$, then $R_{s,t}(G(x)) = \infty$.
3. If G consists of subgraphs G_1 and G_2 connected in series as described above, then $R_{s,t}(G(x)) = R_{s,t}(G_1(x)) + R_{s,t}(G_2(x))$.
4. If G consists of subgraphs G_1 and G_2 connected in parallel as described above, then $(R_{s,t}(G(x)))^{-1} = (R_{s,t}(G_1(x)))^{-1} + (R_{s,t}(G_2(x)))^{-1}$.
5. If $G(x)$ is a subgraph of $G(y)$, then $R_{s,t}(G(x)) \geq R_{s,t}(G(y))$.
6. If s and t are connected in $G(x)$, then $R_{s,t}(G(x)) \leq d$, where d is the length of the shortest path from s to t.

Effective Capacitance

1. If G consists of a single edge between s and t, and $G(x)$ does not include the edge $\{s,t\}$, then $C_{s,t}(G(x)) = 1$.
2. If s and t are connected in $G(x)$, then $C_{s,t}(G(x) = \infty$.
3. If G consists of subgraphs G_1 and G_2 connected in series as described above, then $(R_{s,t}(G(x)))^{-1} = (R_{s,t}(G_1(x)))^{-1} + (R_{s,t}(G_2(x)))^{-1}$.
4. If G consists of subgraphs G_1 and G_2 connected in parallel, then $R_{s,t}(G(x)) = R_{s,t}(G_1(x)) + R_{s,t}(G_2(x))$.
5. If $G(x)$ is a subgraph of $G(y)$, then $C_{s,t}(G(x)) \leq C_{s,t}(G(y))$.
6. If s and t are not connected in $G(x)$, $C_{s,t}(G(x))$ is less than the size of the smallest cut in G between s and t.

We analyze our algorithms using these properties rather than first principles to demonstrate the relative ease of bounding the query complexity of span program algorithms for st-connectivity problems.

Now we can describe the performance of the span program algorithm for deciding st-connectivity:

▶ **Theorem 4.** *[10] Let $G = (V, E)$ be a graph with $s, t \in V(G)$. Then there is a span program algorithm whose bounded-error quantum query complexity of evaluating whether s and t are connected in $G(x)$ promised $x \in X$ and $X \subseteq \{0,1\}^N$ is*

$$O\left(\sqrt{\max_{\substack{x\in X \\ R_{s,t}(G(x))\neq\infty}} R_{s,t}(G(x)) \times \max_{\substack{x\in X \\ C_{s,t}(G(x))\neq\infty}} C_{s,t}(G(x))}\right). \tag{1}$$

Furthermore, the space complexity is

$$O(\max\{\log(|E|), \log(|V|)\}), \tag{2}$$

and the time complexity is U *times the query complexity, where* U *is the time required to perform one step of a quantum walk on G (see [11]).*

Ito and Jeffery describe an algorithm that can be used to estimate $R_{s,t}(G(x))$. It depends on a quantity called the negative witness size, which we denote $\widetilde{R}_-(x, G)$ (this is the quantity $\widetilde{w}_-(x)$ of [9] tailored to the case of the st-connectivity span program). Let $\mathcal{L}(U, \mathbb{R})$ be the set of linear maps from a set U to $\mathbb{R}$. Then we have the following definition:

▶ **Definition 5** (See Theorem 4.2, [9]). *Let $G = (V, E)$ with $s, t \in V$. If $G(x)$ is connected from s to t, let $V_x \subseteq V$ be the set of vertices connected to both s and t. Then there is a unique map $\mathcal{V}_x \in \mathcal{L}(V_x, \mathbb{R})$ such that $\mathcal{V}_x(s) = 1$, $\mathcal{V}_x(t) = 0$, and $\sum_{\{u,v\}\in E(G(x))} (\mathcal{V}_x(u) - \mathcal{V}_x(v))^2$ is minimized. Then the negative approximate witness size of input x on the graph G is*

$$\widetilde{R}_-(x, G) = \min_{\mathcal{V}\in\mathcal{L}(V,\mathbb{R}):\mathcal{V}(u)=\mathcal{V}_x(u) \text{ if } u\in V_x} \sum_{\{u,v\}\in E} (\mathcal{V}(u) - \mathcal{V}(v))^2 . \tag{3}$$

▶ **Theorem 6.** *Let G be a graph with $s, t \in V(G)$. Then the bounded-error quantum query complexity of estimating $R_{s,t}(G(x))$ to multiplicative error ϵ promised s and t are connected in $G(x)$ and $x \in X$ for $X \subseteq \{0,1\}^N$ is $\widetilde{O}\left(\epsilon^{-3/2}\sqrt{R_{s,t}(G(x))\widetilde{R}_-}\right)$, where $\widetilde{R}_- = \max_{x\in X} \widetilde{R}_-(x, G)$.*

3 Quantum Algorithms for Detecting and Characterizing Cycles

In this section, we prove the following results on detecting and characterizing cycles:

▶ **Theorem 7.** *Let G be the complete graph on n vertices. If we are promised that either $G(x)$ is connected with circuit rank at least r, or $G(x)$ is not connected and contains at most μ edges, then the bounded-error quantum query complexity of detecting a cycle in $G(x)$ is $O(\mu\sqrt{n/r})$.*

▶ **Theorem 8.** *Given a generic graph G with m edges, and a parameter $\epsilon \ll 1$ (here ϵ can be a constant or can depend on the input) there is a quantum algorithm that estimates the circuit rank r of $G(x)$ to multiplicative error ϵ using $\widetilde{O}\left(\epsilon^{-3/2}\sqrt{m\mu/r}\right)$ applications of O_x, under the promise that $G(x)$ has at most μ edges.*

A few notes on these theorems:

- Theorem 7 has a worst case upper bound of $O(n^{3/2})$, which matches the optimal lower bound. This is because $r \geq 1$ (r takes value 1 in the case of a single cycle) and $\mu \leq n - 1$ (since a graph without cycles must be a forest).
- In Theorem 8, if r and ϵ are $O(1)$ and if nothing is known about μ (in which case it could be as large as m), one would need to query all edges of the graph. However, given a promise that r is large, for example if $r = \Omega(m^\beta)$ for a positive constant β, or a promise on μ, we can do better than the trivial classical algorithm of querying all edges.

We prove both of these results using a reduction from cycle detection to *st*-connectivity. Specifically we construct a graph G_{cyc} such that $G_{\text{cyc}}(x)$ has an *st*-path if and only if $G(x)$ has a cycle. We note that there is a cycle in $G(x)$ if and only if an edge $\{u, v\}$ is present in $G(x)$ and there is a path from u to v in $G(x)$ that does not use the edge $\{u, v\}$. Thus, our reduction tests every edge in G to determine whether these two conditions are satisfied. We use the encoding of logical AND and OR into *st*-connectivity using serial and parallel composition, as described in Section 2 and in Refs. [14, 11].

We now describe how to build up G_{cyc} from simpler graphs. For an edge $\ell = \{u, v\} \in E(G)$ let G_ℓ^- be the graph that is the same as G, except with the edge ℓ removed, and the vertex u labeled as s, and the vertex v labeled as t. (The choice of which endpoint of ℓ is s and which is t is arbitrary - either choice is acceptable). Each edge in G_ℓ^- is associated with the same literal of x as the corresponding edge in G. Thus there is an *st*-path in $G_\ell^-(x)$ if and only if there is a path between u and v in $G(x)$ that does not go through $\{u, v\}$.

Next, for an edge $\ell \in E(G)$ let G^1_ℓ be the graph with exactly two vertices labeled s and t, and one edge between them. The one edge in G^1_ℓ is associated with the same literal bit of x as ℓ. Thus there is an st-path in $G^1_\ell(x)$ if and only if $\ell \in E(G(x))$.

Next, we create the graph G_ℓ by connecting G^1_ℓ and G^-_ℓ in series, while leaving the associations between edges and literals the same. Then because connecting st-connectivity graphs in series is equivalent to logical AND, there is an st-path through $G_\ell(x)$ if and only if there is a cycle in $G(x)$ passing through ℓ.

Finally, we create the graph G_{cyc} by connecting all of the graphs G_ℓ (for each $\ell \in E(G)$) in parallel, again retaining the association between edges and literals. Since attaching graphs in parallel is equivalent to logical OR, $G_{\text{cyc}}(x)$ has an st-path if and only if there is a cycle through some edge of $G(x)$. See Figure 1 for an example of the construction of G_{cyc}.

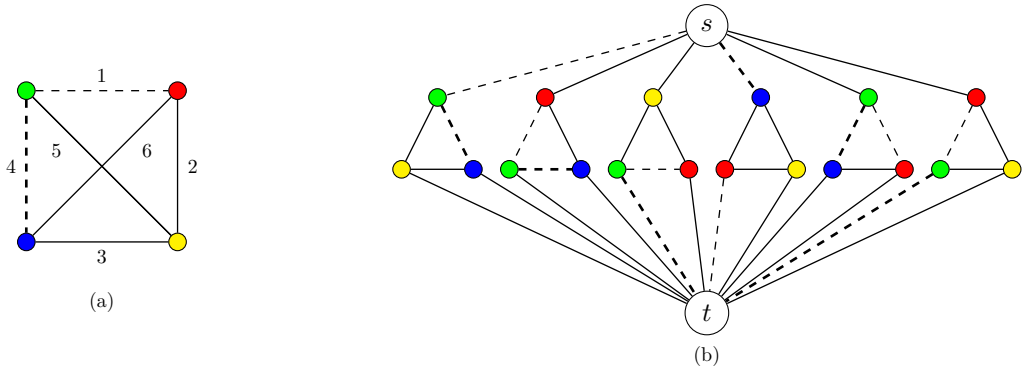

Figure 1 (a) A graph $G(x)$, where edges 2, 3, 5, and 6 are present (solid lines indicate the presence of an edge, dashed lines indicate the absence of an edge). (b) The graph $G_{\text{cyc}}(x)$ that $G(x)$ produces. There is a cycle involving the edges 2, 3, and 6 in $G(x)$, and thus there are paths from s to t in the subgraphs G_2, G_3, and G_6 in $G_{\text{cyc}}(x)$.

In order to use Theorem 4 to determine the query complexity of deciding st-connectivity on $G_{\text{cyc}}(x)$, we next analyze the effective resistance (respectively capacitance) of $G_{\text{cyc}}(x)$ in the presence (resp. absence) of cycles in $G(x)$. We first show the following relationship between effective resistance and circuit rank:

▶ **Lemma 9.** *Let r be the circuit rank of $G(x)$. Then*

$$R_{s,t}(G_{cyc}(x)) = \frac{1}{r}. \tag{4}$$

The proof of Lemma 9 uses the following result from Ref. [5] relating effective resistance and spanning trees:

▶ **Theorem 10** ([5])**.** *Let $\{u, v\} = \ell$ be an edge in a connected graph G. Then the effective resistance between vertices u and v is equal to the number of spanning trees that include an edge ℓ, divided by the total number of spanning trees:*

$$R_{u,v}(G) = \frac{t_\ell(G)}{t(G)}. \tag{5}$$

Proof of Lemma 9. Using the rules that the effective resistance of graphs in series adds, (Effective Resistance Property 3), and the inverse effective resistance of graphs in parallel

adds, (Effective Resistance Property 4), we have:

$$R_{s,t}(G_{\text{cyc}}(x)) = \left(\sum_{(u,v)\in E(G(x))} 1 - R_{u,v}(G(x)) \right)^{-1}. \tag{6}$$

(We include this relatively straightforward calculation in Appendix A.)

We next relate the righthand side of Equation (6) to the circuit rank. Let $G(x)$ be a graph with κ connected components. Let $g_i(x)$ be a subgraph consisting of the i^{th} connected component of G, with n_i vertices. We count the number of times edges are used in all spanning trees of $g_i(x)$ in two ways. First, we multiply the number of spanning trees by the number of edges in each spanning tree. Second, for each edge we add the number of spanning trees that include that edge. Setting these two terms equal, we have,

$$t(g_i(x))(n_i - 1) = \sum_{\ell\in E(g_i(x))} t_\ell(g_i(x)). \tag{7}$$

Rearranging, and using Theorem 10 we have

$$n_i - 1 = \sum_{\ell\in E(g_i(x))} \frac{t_\ell(g_i(x))}{t(g_i(x))} = \sum_{\{u,v\}\in E(g_i(x))} R_{u,v}(g_i(x)), \tag{8}$$

where if the sum has no terms (i.e. $E(g_i(x)) = \varnothing$), we define it to be zero.

Summing over all κ components of $G(x)$, we have

$$n - \kappa = \sum_{\{u,v\}\in E(G)} R_{u,v}(G(x)), \tag{9}$$

Finally, using the fact that

$$\sum_{\{u,v\}\in E(G(x))} 1 = m, \tag{10}$$

where m is the number of edges in $G(x)$, and combining with Equation (9), and Definition 1, we have

$$\sum_{\{u,v\}\in E(G(x))} 1 - R_{u,v}(G(x)) = r. \tag{11}$$

Finally Equation (11) and Equation (6) give the result. ◀

We next analyze the effective capacitance of $G_{\text{cyc}}(x)$ in the case of no cycles in $G(x)$:

▶ **Lemma 11.** *If G is the complete graph on n vertices and $G(x)$ has no cycles and at most μ edges, then, $C_{s,t}(G_{cyc}(x)) = O(n\mu^2)$.*

Proof. We first analyze the effective capacitance of the subgraph $G_\ell(x)$ in two cases, when $\ell \in E(G(x))$ and when $\ell \notin E(G(x))$.

When $\ell \in E(G(x))$, then using Effective Capacitance Properties 2 and 3, we have $C_{s,t}(G_\ell(x)) = C_{s,t}(G_\ell^-(x))$. Then using Effective Capacitance Property 6, we have that $C_{s,t}(G_\ell^-(x))$ is less than the size of the cut between vertices s and t in $G_\ell^-(x)$. Since there are μ edges and n vertices in $G(x)$, this quantity is bounded by $O(n\mu)$. (The worst case is when there are $\Omega(\mu)$ vertices connected to s, e.g.)

When $\ell \notin E(G(x))$, then using Effective Capacitance Properties 1 and 3, $C_{s,t}(G_\ell(x)) = O(1)$.

Since there are $n-\mu$ graphs $G_\ell(x)$ with $\ell \notin E(G(x))$ and μ graphs $G_\ell(x)$ with $\ell \in E(G(x))$, using Effective Capacitance Property 4 for graphs connected in parallel, we have that $C_{s,t}(G_{\text{cyc}}(x)) = O(\mu^2 n)$. ◀

In order to prove Theorem 8, we need to analyze $\widetilde{R}_-(x, G_{\text{cyc}})$:

▶ **Lemma 12.** *For a graph G with m edges, let $X = \{x : G(x) \text{contains a cycle and } |E(G(x))| \leq \mu\}$ and let $\widetilde{R}_- = \max_{x \in X} \widetilde{R}_-(x, G_{cyc})$. Then $\widetilde{R}_- = O(m\mu)$.*

Proof. Looking at Equation (3), for $\ell \notin E(G(x))$, we have that all $v \in V(G_\ell)$ (except s and t) are not in V_x, so any choice of $\mathcal{V}$ on these vertices will give an upper bound on the minimizing map. We choose $\mathcal{V}(v) = 0$ for these vertices to give us our bound, which contributes 1 to the sum for each such subgraph. Thus edges in these subgraphs contribute $m - \mu$ to the total.

For $\ell \in E(G(x))$, for vertices in these subgraphs which are also part of V_x, they will get mapped by $\mathcal{V}_x$ to values between 0 and 1 inclusive (since $\mathcal{V}_x$ can be seen as the voltage induced at each point by a unit potential difference between s and t). If we choose the remaining vertices to also get mapped to values between 0 and 1 by $\mathcal{V}$, we will again have an upper bound on the minimum. Then $(\mathcal{V}(u) - \mathcal{V}(v))^2 \leq 1$ across all edges in these subgraphs. Since there are m edges in each subgraph, and μ such subgraphs, edges in these subgraphs contributes $m\mu$ to the total.

Combining the two terms, we have that $\widetilde{R}_-(x, G_{\text{cyc}}) \leq m - \mu + m\mu = O(m\mu)$. ◀

Now we can put these results together to prove Theorem 8:

Proof of Theorem 8. Using Theorem 6, Lemma 9, and Lemma 12, we can estimate one over the circuit rank (i.e. $1/r$) to multiplicative error ϵ. That is, we get an estimate of $1/r$ within $(1 \pm \epsilon)/r$. Now if we take the inverse of this estimate, we get an estimate of of r within $r/(1 \pm \epsilon)$. But since $\epsilon \ll 1$, taking the Taylor expansion, we have $1/(1 \pm \epsilon) \approx (1 \pm \epsilon)$ to first order in ϵ. ◀

4 Algorithms for Detecting Odd Paths, Bipartiteness, and Even Cycles

In this section, we note that a slight variation on one of the st-connectivity problems considered by Cade et al. in Ref. [7] can be used to detect odd paths and bipartiteness; furthermore the bipartiteness testing algorithm we describe is optimal in query complexity, and far simpler than the bipartiteness algorithm in [7]. We then use a similar construction to create a reduction from even-length cycle detection to *st*-connectivity.

All of these algorithms involve the bipartite double graph of the original graph. Given a graph G with vertices u and v, let $K^G_{u,v}$ be the bipartite double graph of G, with vertex u_0 relabeled as s, and vertex v_1 relabeled as t. To define $K^G_{u,v}(x)$, if $\{x, y\} \in E(G)$ is associated with a literal, then $\{x_0, y_1\}$ and $\{x_1, y_0\}$ in $E(K^G_{u,v})$ are associated with the same literal.

We first show a reduction from detecting an odd-length path to *st*-connectivity on K^G:

▶ **Lemma 13.** *Let G be a graph with vertices u and v. There is an odd-length path from u to v in $G(x)$ if and only if there is an st-path in $K^G_{u,v}(x)$*

Proof. Suppose there is an odd-length path from u to v in $G(x)$. Let the path be $u, \eta^1, \eta^2, \ldots, \eta^k, v$ where k is an even integer greater than or equal to 0. Then there is a path $s, \eta^1_1, \eta^2_0, \ldots, \eta^k_0, t$, in $K^G_{u,v}(x)$ (where the path goes through $\eta^i_{i \bmod 2}$). For the other direction, if there is a path from s to t in $K^G_{u,v}(x)$, there is an odd-length path from u to v in G. Note that any path in $K^G_{u,v}(x)$ must alternate between 0- and 1-labeled vertices. If there is a path that starts at a 0-labeled vertex and ends at a 1-labeled vertex, it must be an odd-length path. Then there must be the equivalent path in $G(x)$, but without the labeling. See Figure 2 for an example of this reduction. ◀

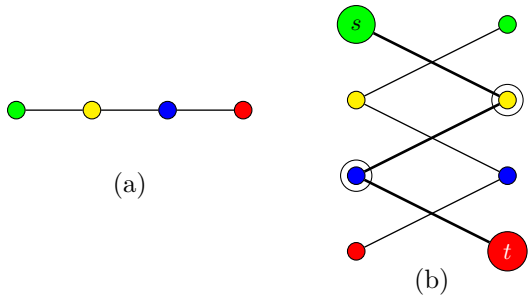

Figure 2 (a) A simple example of a graph G with an odd-length path between green and red vertices. (b) The bipartite double graph $K^G_{green,red}$ with a path between s and t.

▶ **Theorem 14.** *Let G be a graph with n vertices and m edges, with vertices u and v. Then there is a bounded-error quantum query algorithm that detects an odd length path from u to v in G using $O(\sqrt{nm})$ queries.*

Proof. Using Lemma 13 we reduce the problem to st-connectivity on $K^G_{u,v}$. Then using Theorem 4, we need to bound the largest effective resistance and effective capacitance of $K^G_{u,v}(x)$ for any string x. The longest possible path from s to t in $K^G_{u,v}(x)$ is $O(n)$ so by Effective Resistance Property 6, $R_{s,t}(K^G_{u,v}(x)) = O(n)$. The longest possible cut between s and t is $O(m)$, so by Effective Capacitance Property 6, $C_{s,t}(K^G_{s,t}(x)) = O(m)$. This gives the claimed query complexity. ◀

Note u_0 is connected to u_1 in $K^G(x)$ if and only if there is an odd-length path from u to itself in $G(x)$, where this path is allowed to double back on itself, as in Figure 3. This odd-length path in turn occurs if and only if the connected component of $G(x)$ that includes u is not bipartite (has an odd cycle)! Thus if we are promised that $G(x)$ is connected, we can pick any vertex in G, run the algorithm of Theorem 14 on $K^G_{u,u}(x)$, and determine if the graph is bipartite, which requires $O(\sqrt{nm})$ queries.

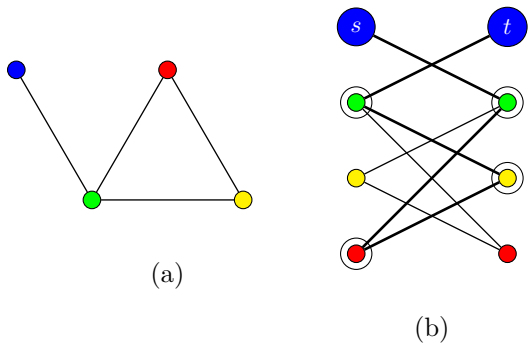

Figure 3 (a) A graph G with an odd cycle. (b) The bipartite double graph K^G with a path between the two green vertices.

On the other hand, if we are not promised that $G(x)$ is connected, we simply need to check whether there is an odd path from any of the n vertices of G to itself, and we now show that doing this check does not increase the query complexity. We use a similar strategy as with cycle detection:

▶ **Theorem 15.** *Let G be a graph with n vertices and m edges. Then there is a bounded-error quantum query algorithm that detects an odd cycle (in effect, non-bipartiteness) in $O(\sqrt{nm})$ queries.*

Proof Sketch. Let G_{bip} be the graph that consists of the the graphs $K^G_{u,u}$ composed in parallel for all $u \in V(G)$. This amounts to evaluating the logical OR of there being an odd cycle connected to any vertex in G.

A similar analysis as in cycle detection shows that the effective resistance of G_{bip} is $O(1)$, if there is an odd cycle. On the other hand, since there are n copies of K^G in this new graph, and each copy has m edges, the largest possible cut is $O(nm)$. Applying Theorem 4 gives the result. ◀

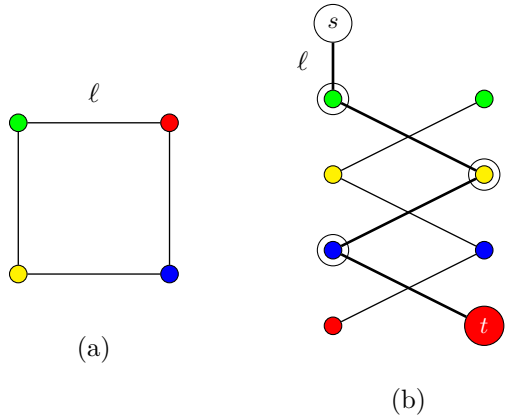

Figure 4 (a) An simple example of a graph G with an even-length cycle. (b) The bipartite double graph $K^G_{green,red}$ connected in series with $G^1_{\{red,green\}}$. We see there is a path from s to t in this graph, corresponding to an even-length cycle passing through ℓ.

Finally, we show how to detect even cycles:

▶ **Theorem 16.** *Let G be a graph with n vertices and m edges. Then there is a bounded-error quantum query algorithm that detects an even-length cycle in $O(\sqrt{nm})$ queries.*

Proof. For an edge $\ell = \{u, v\} \in E(G)$, note that there is an st-path in $K^{G^-_\ell}_{u,v}$ if and only if there is an odd-length path from u to v that does not use the edge ℓ itself. Thus if we consider the graph composed of G^1_ℓ and $K^{G^-_\ell}_{u,v}$ in series, which we denote G^E_ℓ, there is an st-path if and only if there is an even-length cycle through ℓ. Finally, if we compose the graphs G^E_ℓ in parallel for all $\ell \in E(G)$, we obtain a graph that has an st-path if and only if there is an even cycle passing through some edge in G, as in Figure 4.

As in our previous analyses of cycle detection and bipartiteness, if there is an even cycle, the effective resistance will be $O(1)$. On the other hand, if there is no even-length cycle, then it is a fairly well known fact that the number of edges in G is $O(n)$. Then similar to previous analyses, for each graph G^E_ℓ such that $\ell \in E(G)$, we have that the cut is $O(m)$. Otherwise, for each graph G^E_ℓ such that $\ell \in E(G)$, we have that the cut is $O(1)$. Thus a bound on the size of the total cut is $O(n^2 + nm) = O(nm)$ (assuming that $n = O(m)$.) Applying Theorem 4 gives the result. ◀

References

1 C. Alvarez and R. Greenlaw. A compendium of problems complete for symmetric logarithmic space. *Computational Complexity*, 9(2):123–145, 2000.

2 A. Āriņš. Span-program-based quantum algorithms for graph bipartiteness and connectivity. In *International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 35–41. Springer, 2015.

3 A. Belovs. Span programs for functions with constant-sized 1-certificates. In *Proceedings of the 44th Symposium on Theory of Computing (STOC 2012)*, pages 77–84, 2012.

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

5 N. Biggs. Algebraic Potential Theory on Graphs. *Bulletin of the London Mathematical Society*, 29, 1997.

6 B. Bollobás. *Modern graph theory*. Springer Science & Business Media, 2013.

7 C. Cade, A. Montanaro, and A. Belovs. TIME AND SPACE EFFICIENT QUANTUM ALGORITHMS FOR DETECTING CYCLES AND TESTING BIPARTITENESS. *Quantum Information and Computation*, 18(1&2):0018–0050, 2018.

8 A. M. Childs and R. Kothari. Quantum query complexity of minor-closed graph properties. *SIAM Journal on Computing*, 41(6):1426–1450, 2012.

9 T. Ito and S. Jeffery. Approximate span programs. *Algorithmica*, pages 1–38, 2015.

10 M. Jarret, S. Jeffery, S. Kimmel, and A. Piedrafita. Quantum Algorithms for Connectivity and Related Problems. In *26th Annual European Symposium on Algorithms (ESA 2018)*, pages 49:1–49:13, 2018. doi:10.4230/LIPIcs.ESA.2018.49.

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

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

13 T. J. McCabe. A complexity measure. *IEEE Transactions on software Engineering*, SE-2(4):308–320, 1976.

14 N. Nisan and A. Ta-Shma. Symmetric Logspace is Closed Under Complement. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing (STOC 1995)*, pages 140–146. ACM, 1995. doi:10.1145/225058.225101.

15 B. W. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 544–551. IEEE, 2009.

16 B. W. Reichardt. Reflections for quantum query algorithms. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 560–569, 2011.

17 R. Yuster and U. Zwick. Finding even cycles even faster. *SIAM Journal on Discrete Mathematics*, 10(2):209–222, 1997.

18 A. Zamora. An Algorithm for Finding the Smallest Set of Smallest Rings. *Journal of Chemical Information and Computer Sciences*, 16(1):40–43, 1976. doi:10.1021/ci60005a013.

19 S. Zhang. On the power of Ambainis lower bounds. *Theoretical Computer Science*, 339(2-3):241–256, 2005.

A Effective Resistance of G_{cyc}

We relate the effective resistance across edges in G to the effective resistance between s and t in G_{cyc}. (The proof also applies to $G(x)$ and $G_{\text{cyc}}(x)$.) We will write $R_{s,t}(G_{\text{cyc}})$ in terms of a sum of $R_{u,v}(G)$ where (u, v, ℓ) is an edge on a cycle in G.

Consider an edge $\{u, v\} \in E(G)$. Then using Effective Resistance Property 4 (for graphs composed in parallel), we have

$$\frac{1}{R_{u,v}(G)} = 1 + \frac{1}{R_{s,t}(G_\ell^-)}. \tag{12}$$

Rearranging, we have

$$R_{s,t}(G_\ell^-) = \frac{R_{u,v}(G)}{1 - R_{u,v}(G)} \tag{13}$$

Then using Effective Resistance Property 3 (for graphs composed in series), we have that

$$\begin{aligned} R_{s,t}(G_\ell) &= R_{s,t}(G_\ell^-) + 1 \\ &= \frac{R_{u,v}(G)}{1 - R_{u,v}(G)} + 1 \\ &= \frac{1}{1 - R_{u,v}(G)}. \end{aligned} \tag{14}$$

Finally, using Effective Resistance Property 4 (graphs composed in parallel) again, we have

$$\frac{1}{R_{s,t}(G_{\text{cyc}})} = \sum_{\{u,v,\}\in E(G)} 1 - R_{u,v}(G). \tag{15}$$