

Quantum Tutte Embeddings

Shion Fukuzawa, Michael T. Goodrich, and Sandy Irani

Dept. of Computer Science, Univ. of California, Irvine, USA

Abstract. Using the framework of Tutte embeddings, we begin an exploration of *quantum graph drawing*, which uses quantum computers to visualize graphs. The main contributions of this paper include formulating a model for quantum graph drawing, describing how to create a graph-drawing quantum circuit from a given graph, and showing how a Tutte embedding can be calculated as a quantum state in this circuit that can then be sampled to extract the embedding. To evaluate the complexity of our quantum Tutte embedding circuits, we compare them to theoretical bounds established in the classical computing setting derived from a well-known classical algorithm for solving the types of linear systems that arise from Tutte embeddings. We also present empirical results obtained from experimental quantum simulations.

Keywords: Tutte embeddings; quantum computing; linear systems.

1 Introduction

Quantum computing studies ways to leverage the principles of quantum mechanics to perform computations; see, e.g., Nielson and Chuang [17]. Unlike classical computing, where bits can only be in one of two states (0 or 1), quantum computing uses quantum bits, or *qubits*, which can exist in multiple states at once, expressed as a linear combination of states with complex coefficients. This property of qubits, called *superposition*, allows quantum computers to perform certain computations faster than what is believed possible with classical computers. Moreover, although quantum computers are still in their infancy and face significant technological challenges, quantum computing holds tremendous promise for advancing fields ranging from chemistry and physics to artificial intelligence and machine learning. At the very least, quantum computing is providing interesting alternatives to classical notions of what is effectively computable.

As is well-known, graphs are used to model a wide range of phenomena, including molecular bonds, social network interactions, biochemical pathways, and computer networks. Fields that study such phenomena include promising applications of quantum computing; see, e.g., [1, 2, 15, 16]. In this paper, we therefore begin an exploration of *quantum graph drawing*, which studies how to use quantum computers to visualize graphs. As a first step in this exploration, we focus in this paper on quantum circuits for what is arguably the first graph

drawing algorithm—*Tutte embeddings* [22].¹ As we review in this paper, a Tutte embedding is a type of force-directed graph drawing paradigm that can be viewed as a simulation of a physical system in which edges are represented as springs. Intuitively, given a graph, G , the goal of this paradigm is to find a layout that minimizes the total energy of the physical system defined by G , after “pinning” some of the vertices in G to specified locations, so as to create an aesthetically pleasing visualization of G ; see, e.g., [7, 14].

Although we focus on Tutte embeddings as a specific graph drawing paradigm, our goal in this paper is to formulate an approach to quantum graph drawing that could potentially be applied to other graph drawing paradigms as well. That is, although our approach uses a quantum linear systems solver to compute the coordinates of the vertices in a graph embedding, we can envision that other types of quantum circuits could be used for other graph drawing paradigms, which could have similar data flows as the methods we employ in this paper.

A quantum linear systems solver computes a vector x that satisfies $Ax = b$ for input matrix, A , and vector, b . The matrix A is embedded in the quantum circuit and the vector b must be prepared as a quantum state. In a Tutte embedding, the matrix A encodes spring-optimization equations for the edges of a graph, so one of the first challenges we explore is how to represent a graph in a quantum circuit. This requires developing a method for effectively encoding a graph in the gates of a quantum circuit, which ideally satisfies some sparsity condition. The output embedding is encoded in the vector x as a quantum state, so another issue that must be addressed is how to extract the drawing from the quantum computer.

Our Contributions. Using the framework of Tutte embeddings, we provide in this paper a proof of concept for quantum graph drawing. So as to avoid introducing additional complications in this endeavor, we have not attempted to fully optimize the performance of the resulting quantum graph-drawing circuit. Instead, the contributions of this paper should be seen as providing a concrete example of an algorithm in quantum graph drawing, including formulating the model and describing how to convert a graph into a quantum graph-drawing circuit, which calculates a Tutte embedding as a quantum state that then can be sampled to extract the embedding. We compare the resulting complexity for our quantum Tutte embedding circuit to theoretical bounds for computing such embeddings in the classical computing setting that can be derived from a well-known classical algorithm for solving the types of linear systems that arise for Tutte embeddings [19], as we review in the next section. In addition, we also provide empirical results from experimental quantum simulations.

¹ Technically, proofs of Fáry’s Theorem, showing that simple, planar graphs have straight-line planar embeddings, came earlier [8, 20, 23], but these proofs, unlike Tutte’s algorithm, don’t provide vertex coordinates.

2 Preliminaries

In order for this paper to be as self-contained as possible, we provide a brief primer on quantum computing in an appendix.

A Brief Review of Tutte Embeddings. In addition, let us provide a brief review of Tutte embeddings [22]. Let $G = (V, E)$ be an n -vertex simple, 3-connected planar graph. Let f be a face of G , which we will consider to be the outer face. A **Tutte embedding** of G is a crossing-free straight-line embedding of G such that the outer face, f , is a convex polygon and such that each interior vertex (not on f) is at the average (or barycenter) of its neighbors' positions. We enforce the first of these properties by “pinning” the vertices of f to be the vertices of a convex polygon, which is typically a regular convex polygon. We enforce the second property by imagining that the edges of G are idealized springs with preferred length 0, so that the minimum-energy configuration of the interior vertices determines their positions. In particular, we can formulate two linear equations for each internal vertex, u , of G as follows (e.g., see the well-known graph-drawing book by Di Battista, Eades, Tamassia, and Tollis [7]):

$$\sum_{v \in N(u)} (x_u - x_v) = 0,$$

and

$$\sum_{v \in N(u)} (y_u - y_v) = 0,$$

where $p_v = (x_v, y_v)$ is the point to which vertex v is assigned, and $N(u)$ denotes the set of neighbors of u in G , i.e., $N(u) = \{v \mid (u, v) \in E\}$. Note that for a vertex, v , on the outer face, f , we pin $p_v = (x_v^*, y_v^*)$; hence, x_v^* and y_v^* are fixed constants. As Tutte showed [22] (and which has been repeated, e.g., by Hopcroft and Kahn [11] and Floater [9]), solving the above linear system of equations produces a planar straight-line embedding of G such that every face is convex.

We subdivide $N(u)$ into the set, $N_I(u)$, of interior-neighbors of u , and $N_f(u)$, neighbors of u on the external-face, f . That is,

$$N_I(u) = \{v \mid (u, v) \in E \text{ and } v \notin f\} \quad \text{and} \quad N_f(u) = \{v \mid (u, v) \in E \text{ and } v \in f\}.$$

Also, let $\deg(u)$ degree of the vertex, u . Then we can rewrite the above linear equations for each interior vertex, u , as follows:

$$\deg(u) x_u - \sum_{v \in N_I(u)} x_v = \sum_{v \in N_f(u)} x_v^*,$$

and

$$\deg(u) y_u - \sum_{v \in N_I(u)} y_v = \sum_{v \in N_f(u)} y_v^*,$$

where $p_v = (x_v^*, y_v^*)$ is the point to which the vertex v is pinned if $v \in f$. Note that the righthand sides of the above equations are 0 for each vertex, u , that is

not adjacent to an outer-face vertex. That is, this set of equations defines two linear systems,

$$Ax = b_x,$$

and

$$Ay = b_y,$$

where x and y are, respectively, the vector of x - and y -coordinates of the points assigned to the interior vertices in G , and the entries of b_x and b_y are 0 except for vertices adjacent to pinned vertices. Note further that the $(n - |f|) \times (n - |f|)$ matrix, A , is symmetric. Moreover, it is closely related to the **graph Laplacian**, L , for G , which is defined as follows (see, e.g., [4, 18]):

$$L = D - M,$$

where D is a diagonal matrix formed by the degrees of the vertices in G and M is the adjacency matrix for G (with the same ordering of vertices as in D). Further, the matrix, A , is **diagonally dominant**, that is, $A_{i,i} \geq \sum_{j \neq i} |A_{j,i}|$, for all i . Also, the number of non-zero entries in A is $O(n)$, since the graph, G , is planar.

Given such a symmetric, diagonally-dominate $O(n) \times O(n)$ matrix, A , with $O(n)$ non-zero entries, and an $O(n)$ -vector, b , as well as an error tolerance, $\varepsilon > 0$, Spielman and Teng [19] provide a classical algorithm that produces a vector, $\tilde{x}$, such that $\|A\tilde{x} - b\| < \varepsilon$ and $\|\tilde{x} - x\| \leq \varepsilon$, where x is the solution to $Ax = b$, in time

$$O\left(n \log^{O(1)} n + n 2^{O(\sqrt{\log n \log \log n})} \log(\kappa/\varepsilon)\right),$$

where κ is the **condition number** of the matrix, A , i.e., the ratio of the largest to the smallest non-zero eigenvalue of A . Thus, although the algorithm is fairly complicated and not commonly used in practice, this provides a theoretical classical algorithm for computing an ε -approximation to the coordinates of a Tutte embedding for G . Incidentally, further improving the asymptotics for solving such linear systems in the classical model is an on-going line of research, with more recent results having similar “near-linear” bounds; see, e.g., [12].

Using this result from the classical setting as a theoretical point of comparison, our interest in this paper is to design a quantum circuit for producing Tutte embeddings, based on a well-known quantum linear system solver [10], whose complexity can also be expressed as a function of n , κ , and ε , as well as Δ , the maximum degree of the vertices in G . We have not attempted, however, to optimize the performance of the quantum linear system solver, as we mention above. The contributions of this paper should instead be seen as providing a first example for an algorithm in quantum graph drawing, including formulating the model and describing how to convert a graph into a quantum graph-drawing circuit. The quantum algorithm calculates a Tutte embedding as a quantum state which is then sampled to extract the embedding. We also provide some empirical results from experimental quantum simulations.

3 A Quantum Algorithm for Tutte Embeddings

This section assumes basic familiarity with quantum computing. For completeness, we provide a primer on quantum computing in an appendix. Quantum algorithms for solving linear systems have been explored heavily since being first introduced by Harrow et al. [10], who designed a quantum algorithm whose running time is exponentially faster in the size of the matrix than classical methods, but with a few tradeoffs that will be discussed later in this section. Their algorithm and the follow-up work largely follow the same structure, whose intuition can be captured by the following. Consider a linear system, $A|x\rangle = |b\rangle$, and recall that the matrix A can be expressed in its eigenbasis as $A = \sum_i \lambda_i |u_i\rangle \langle u_i|$. The inverse of A can then be expressed as $A^{-1} = \sum_i \lambda_i^{-1} |u_i\rangle \langle u_i|$, and the vector $|b\rangle$ can also be written in this basis as $|b\rangle = \sum_i b_i |u_i\rangle$. Using this, we can express $|x\rangle$ also in the eigenbasis of A as follows:

$$|x\rangle = A^{-1} |b\rangle \quad (1)$$

$$= \sum_i \frac{\beta_i}{\lambda_i} |u_i\rangle. \quad (2)$$

The objective of the quantum linear system solver is to prepare a state $|x\rangle$ which is proportional to the output state above. A vector corresponding to a quantum state must be normalized so that its L_2 norm is 1, which is why the output is only proportional to $|x\rangle$ given above, which is not necessarily normalized.

Besides the dimensionality, N , of the matrix and the desired precision of the solution, ε , linear system solvers both in the classical and quantum literature critically depend on two other parameters, namely the sparsity, s , and the condition number, κ . A matrix is *s-sparse* if it has at most s nonzero entries in each row, and the condition number is defined to be the ratio between the largest and smallest eigenvalue of A , i.e., $\kappa = \lambda_{\max}/\lambda_{\min}$. Intuitively, this captures how “invertible” the matrix is, as a noninvertible matrix would have $\kappa = \infty$. The following theorem captures the performance of the quantum linear solver algorithm. While there have been subsequent developments in quantum linear systems solvers [3], they do not provide any advantage over the original algorithm for the linear systems we are considering. For a comprehensive survey on quantum linear systems solvers, we refer the reader to Dervovic *et al.* [6].

Theorem 1 (Harrow et al. [10]). *Let A be a Hermitian s -sparse $N \times N$ matrix with condition number κ and let b be a unit vector. Then there is a quantum algorithm that can prepare the state, $|x\rangle$, satisfying the equation*

$$Ax = b \quad (3)$$

in $\tilde{O}(\kappa T_B + \log(N) s^2 \kappa^2 / \varepsilon)$ time. The $\tilde{O}$ suppresses the more slowly growing terms of $(\log^(N))^2$, $\exp\left(O\left(1/\sqrt{\log(t_0, \varepsilon_H)}\right)\right)$, and $\text{polylog}(T/\varepsilon_\Psi)$. Here, T_B is the time required to prepare the input vector $|b\rangle$, ε_Ψ and ε_H are error terms*

accrued in the phase estimation subroutine, and T is usually selected to be $O(\log(N)s^2t_0)$. The ε is the additive error achieved in the output state $|x\rangle$.

Once the output state $|x\rangle$ is prepared, for any Hermitian operator, M , the expected value $x^T M x$ can be measured.

As described above in section 2, the matrix A that appears in the Tutte embedding system is diagonally dominant, and for a fixed degree graph has constant sparsity, s . It is easy to show that this matrix is sparse when the degrees of the vertices are bounded by Δ , for example. The analysis of the condition number is more involved, and we provide an experimental analysis of how the condition number can scale for various classes of graphs.

Our quantum graph drawing algorithm has 3 main components: preparing the input state $|b\rangle$, the procedure for generating the solution vector $|x\rangle$, and finally the measurement of the expected value of an operator M given the state $|x\rangle$. The following is an overview of the algorithm, and specific parts of it will be detailed in the following subsections.

1. Preparation
 - Prepare the input vector $|b\rangle$.
 - Generate a quantum circuit from a sparse representation of A .
2. Run eigenvalue estimation
3. Perform conditional rotations
4. Invert the eigenvalue estimation
5. Repeat steps 2, 3, 4 until success
6. Use output vector $|x\rangle$ to measure summary statistics about an operator M

3.1 Preparing the Vector $|b\rangle$

Recall that for an n -vertex planar graph, we can draw the graph by solving two system of equations, $F(u) = \sum_{(u,v) \in E} (v - u)$, for the x and y coordinates, respectively. Also recall that if k vertices are pinned down, the matrix A representing the graph is close to the graph Laplacian. Thus, the final linear systems will be represented by a system of size $n - k$ by $n - k$. In the literature, such matrices have been defined and studied as “grounded Laplacians.” For example, Pirani et al. [18] show that for weighted d -regular graphs with one pinned vertex, the smallest eigenvalue is $\Theta(1/n)$. We empirically study the condition number for grounded Laplacians of several classes of graphs and present the results below in section 4.

In this paper, we focus on the case where there is a triangular outer face of the graphs, which will always be drawn such that it bounds the rest of the graph. Furthermore, to simplify the input vector, $|b\rangle$, we add a dummy outer face such that the true outer face is connected to this dummy outer face, as is shown in Figure 1. In our construction of the linear system, we pin down this dummy outer face to the coordinates, $(0, 0)$, $(1, 0)$, $(0, 1)$. For the purpose of illustration, when indexing the vertices of the graph we will always index the vertices of the outer face first in a clockwise order starting from the bottom left vertex. When

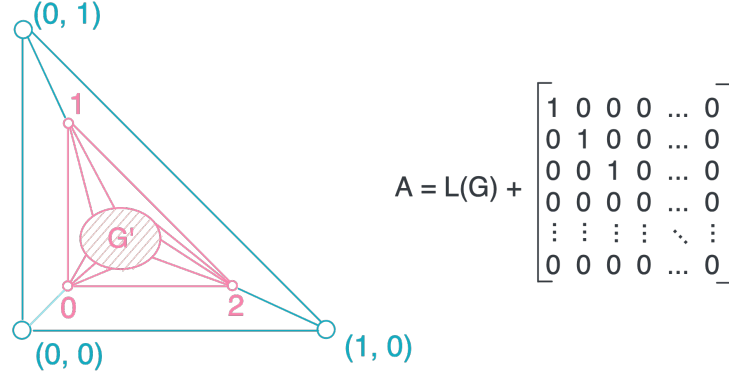


Fig. 1: To simplify the linear system we are solving, we construct our system by creating a dummy outer face drawn in blue, and pinning the three blue vertices to the coordinates $(0,0)$, $(0,1)$, and $(1,0)$. Under this construction, if we index the vertices of our graph G such that the three outer vertices are indexed as above, the construction of the matrix A is simply the graph Laplacian $G(L)$ plus a diagonal matrix such that the first three entries are 1, and the remaining are all 0. This indexing scheme also ensures that $|b\rangle$ has a single index with a 1 and 0 everywhere else.

using this indexing scheme, the $|b\rangle$ vector for the x -coordinate system is always the vector with a 1 in index 2 and 0's everywhere else, and the $|b\rangle$ vector for the y -coordinate system is always the vector with a 1 in index 1 and 0's everywhere else.

3.2 Preparing the circuit to implement the matrix A

Recall that quantum gates are unitary operators, and the input matrices are Hermitian. Given a Hermitian matrix, $A = \sum_i \lambda_i |u_i\rangle\langle u_i|$, if we take the matrix exponent of A , then we get an operator U which is unitary:

$$U := e^{-iAt} = \sum_i e^{-i\lambda_i t} |u_i\rangle\langle u_i|. \quad (4)$$

The factor of i is critical in enforcing unitarity, and the constant multiple of t is a parameter we vary during the algorithm. An important preparation step in running this algorithm on a quantum computer is to generate the circuit that implements U .

In the literature, this problem is referred to as Hamiltonian simulation (See Dervovic *et al.* [6] for a more in depth tutorial). The main method we use begins by decomposing our operator into a sum of simpler matrices as $A = \sum_{j=1}^s A_j$, where simple refers to a class of matrices that are known to be easy to implement on a quantum computer. Given this decomposition, equation 4 can be rewritten,

$$e^{-iAt} = e^{-i \sum_{j=1}^s A_j t}. \quad (5)$$

We want to combine smaller circuits where each circuit depends only on a single A_j , but in general matrix products do not commute so we cannot just implement the circuit $\prod_{j=1}^s e^{iA_j t}$. Instead, the Lie-Trotter product theorem [21] states that

$$e^{-i(A+B)t} = \lim_{m \rightarrow \infty} \left(e^{-iAt/m} e^{-iBt/m} \right)^m \quad (6)$$

which has driven many results in Hamiltonian simulation. For the simplest model of approximating equation (6), it is now known that selecting $m = O((\nu t)^2/\varepsilon)$ achieves

$$\left\| \left(e^{-iAt/m} e^{-iBt/m} \right)^m - e^{-i(A+B)t} \right\| \leq \varepsilon, \quad (7)$$

where $\nu := \max\{\|A\|, \|B\|\}$ and $\|\cdot\|$ denotes the spectral norm of the operator.

To decompose the operator A into a sum of terms, we can first express it as the sum of the diagonal D and non-diagonal M which is just the adjacency matrix of the free vertices.

A simple circuit construction for the diagonal matrix e^{-iDt} can be realized by the following. This operation will use two quantum registers, the index register with $\log N$ qubits and a memory register using $\log s$ qubits, where s is the sparsity. The circuit begins by setting the index register into superposition using $\log N$ Hadamard gates. Then for each index, it will perform a $\log N$ qubit Toffoli gate up to $\log s$ times to store the state $|a, d(a)\rangle$ where $d(a)$ is the diagonal element in the matrix of the a -th index. After this step, we apply phase gates parameterized by $-2^i t$ on the i -th qubit. After this step, we then uncompute the memory register by reversing the Toffoli gates. The full transformation of the above circuit is described by the following sequence of equations:

$$|a, 0\rangle \rightarrow |a, d(a)\rangle \quad (8)$$

$$\rightarrow e^{-id(a)t} |a, d(a)\rangle \quad (9)$$

$$\rightarrow e^{-id(a)t} |a, 0\rangle \quad (10)$$

$$= e^{-iDt} |a\rangle |0\rangle \quad (11)$$

Now that we have a circuit computing the diagonal part, what remains is to build a circuit to implement the exponentiated adjacency matrix, M . Unfortunately, M on its own may not be easily translated into a quantum circuit, so this must also be expressed as a sum of terms that can be easily diagonalized. For an adjacency matrix, this can be accomplished by computing an edge coloring on the graph, and decomposing the adjacency matrix into a sum of terms corresponding to each color. Since the graphs we are considering have a maximum degree of Δ , a greedy edge coloring scheme provides a coloring of at most 2Δ terms. After this step, each color, c , corresponds to a 1-sparse matrix that is still Hermitian, meaning that it can easily be diagonalized to be written as $M_c = U_c D_c U_c^\dagger$ where U_c is unitary and D_c is diagonal. Using the identity,

$$e^{-iU H U^\dagger t} = U e^{-iH t} U^\dagger, \quad (12)$$

for conjugations of Hermitian matrices with unitary matrices, we can build the circuit M_c by first constructing the circuit to simulate U_c , then performing the diagonal part the same way as it was done for D .

In the following sections, we will use U to refer to the circuit that implements the approximation to the e^{-iAt} equation decomposed into a sum of a diagonal matrix and the decomposition of the off-diagonal matrix corresponding to the edge coloring, using the Trotterization scheme mentioned in equation (7).

3.3 Solving the Linear System

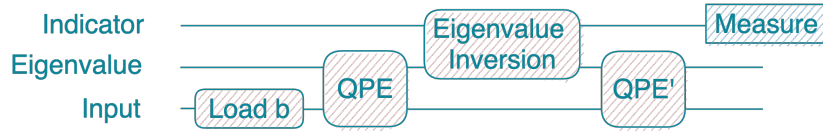


Fig. 2: A high level overview of Harrow's quantum algorithm for solving linear systems. The shaded boxes describe subroutines that will be described in further detail in this section.

In this section, we provide a high-level overview of how Harrow et al.'s linear system solver algorithm [10] works. The first key subroutine is what is referred to in the literature as phase estimation or eigenvalue estimation [13, 17]. Since phase estimation is a standard quantum algorithm, we just describe the input/output behavior of the algorithm and how it is used for linear system solving.

The ideal phase estimation algorithm takes as input a unitary U and one of its eigenvectors $|\psi\rangle$, and the output will be an r -bit approximation of $2^r\theta$ for $e^{2\pi i\theta}$, the eigenvalue corresponding to the input eigenvector. The procedure is described by the equation,

$$|0\rangle_r |\psi\rangle \rightarrow |2^r\theta\rangle |\psi\rangle. \quad (13)$$

The behavior of the algorithm on a generic quantum state $|b\rangle$ can be understood by expressing $|b\rangle$ in the eigenbasis for U . Let $|\psi_j\rangle$ denote the j -th eigenvector of U . Then $|b\rangle$ can be expressed as $|b\rangle = \sum_j \beta_j |\psi_j\rangle$ for some set of coefficient $\{\beta_j\}_j$. Performing the full phase estimation using this state as the input state now prepares a superposition of r -bit approximations of $2^r\theta_j$ where $e^{2\pi i\theta_j}$ is the eigenvalue of $|\psi_j\rangle$ weighted by the coefficient β_j :

$$|0\rangle_r |b\rangle = \sum_j \beta_j |0\rangle_r |\psi_j\rangle \rightarrow \sum_j \beta_j |2^r\theta_j\rangle |\psi_j\rangle \quad (14)$$

Note that $2\pi\theta_j = \lambda_j$ for all j , and a uniform scaling of the eigenvalues will not effect the description of the algorithm below, which is expressed in terms of the

λ_j 's. Once we've prepared the register storing a superposition of the eigenvalues, we use these to prepare the following state:

$$\sum_j \beta_j |\lambda_j\rangle |\psi_j\rangle \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \sqrt{\frac{C}{\lambda_j}} |1\rangle \right), \quad (15)$$

where C is a constant that should be chosen such that $|C| < \lambda_{min}$. This rotation can be performed by preparing a single qubit register initialized in the $|0\rangle$ state, then for each k -th significant bit in the eigenvalue register, perform a controlled rotation by an angle of $\frac{C}{2^k}$. Provided a sufficiently good approximation of the eigenvalue, this will put the quantum device in the state described in equation (15).

Following the controlled rotation, we now do the inverse of the phase estimation procedure, effectively *uncomputing* the eigenvalue register. We have no entanglement with the eigenvalue register now, so we can ignore it for the remaining analysis. The state we have now is

$$\sum_j \beta_j |\psi_j\rangle \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \sqrt{\frac{C}{\lambda_j}} |1\rangle \right), \quad (16)$$

and the final step before preparing the $|x\rangle$ vector is to measure the last register. If we measure $|1\rangle$ in the final register, we end up with the state

$$\sqrt{\frac{1}{\sum_j C^2 |\beta_j|^2 / |\lambda_j|^2}} \sum_j \frac{\beta_j C}{\lambda_j} |\psi_j\rangle, \quad (17)$$

which is exactly the state $|x\rangle$ described in equation (2) up to a normalization factor. The final running time including the time required to perform the matrix to circuit conversion is summarized in the following theorem.

Theorem 2. *Let G be a 3-connected planar graph with n vertices and at most Δ edges per vertex. Let A be the grounded Laplacian and $|b_x\rangle, |b_y\rangle$ be the input vectors for the x - and y - coordinate systems as described in section 3.1. Then, the quantum computer can prepare the normalized solution vector $|x\rangle$ in time $\tilde{O}(sn + \kappa T_B + \kappa^2 s^2 \log(n)/\varepsilon)$, where the first term is the time required to construct the quantum circuit from the input matrix.*

4 Experiments

This section is divided into two sections, the first being an empirical study on the condition number of grounded Laplacian matrices, and the second section being a demonstration of the results obtained from using the quantum algorithm for graph drawing.

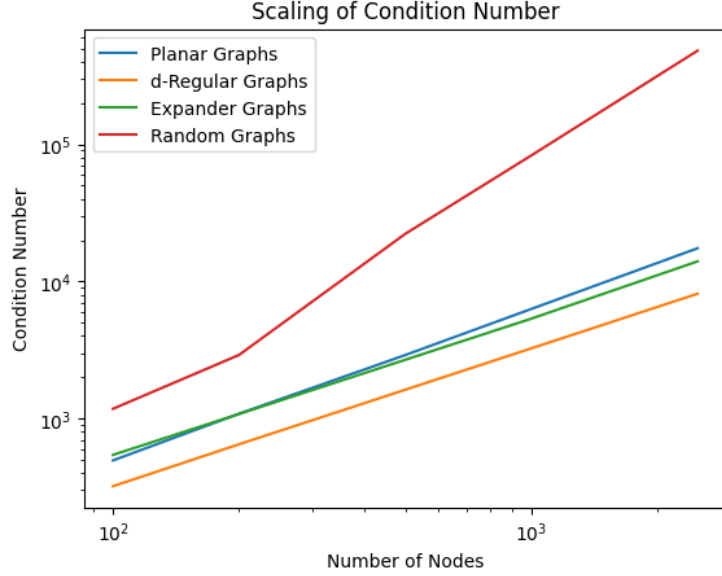


Fig. 3: An empirical study of the scaling of condition numbers for standard classes of graphs. Graphs were generated randomly within each class, and the average is plotted over 100 runs per data point.

4.1 Condition Number

As the quantum algorithm critically relies on the condition number, we begin the exploration with an empirical study of these values for a few classes of graphs. The planar graphs were generated using the technique outlined in the following subsection. The remaining three types of graphs were generated using the built-in graph generation functions for Python’s NetworkX library. The expander graphs are Margulis-Gabber-Galil graphs and the random graphs are Erdős-Rényi graphs. For each instance, the average over 100 samples is plotted in Figure 3. We see in the results that for all chosen classes of graphs, the condition number appears to scale at least linearly.

4.2 Example Quantum Tutte Embeddings

To demonstrate the proof of concept for our Tutte embedding procedure using quantum circuits, we used a public implementation ² of Harrow’s algorithm, which is endorsed by and works with QISKit [5], the leading software package for writing quantum circuits. We note that the off-the-shelf implementation provided here does not perform the optimal matrix-to-circuit construction as outlined in section 3. However, we adopt this version for the purpose of demonstration.

² https://github.com/anedumla/quantum_linear_solvers/tree/main

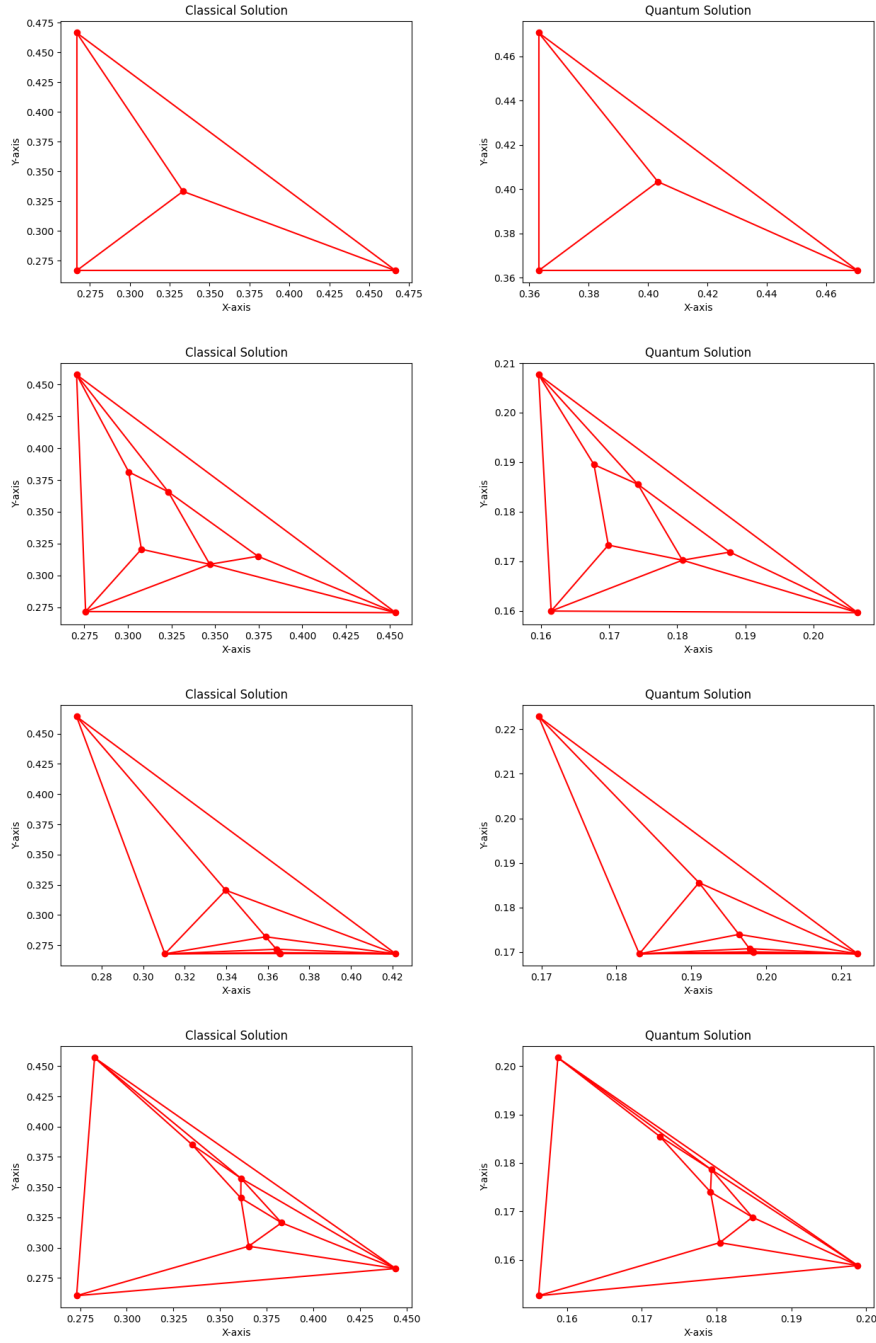


Fig. 4: Graph drawing results using both the classical and quantum solver. The first is drawing a random planar graph with 4 nodes, and the last three are drawing random planar graphs with 8 nodes.

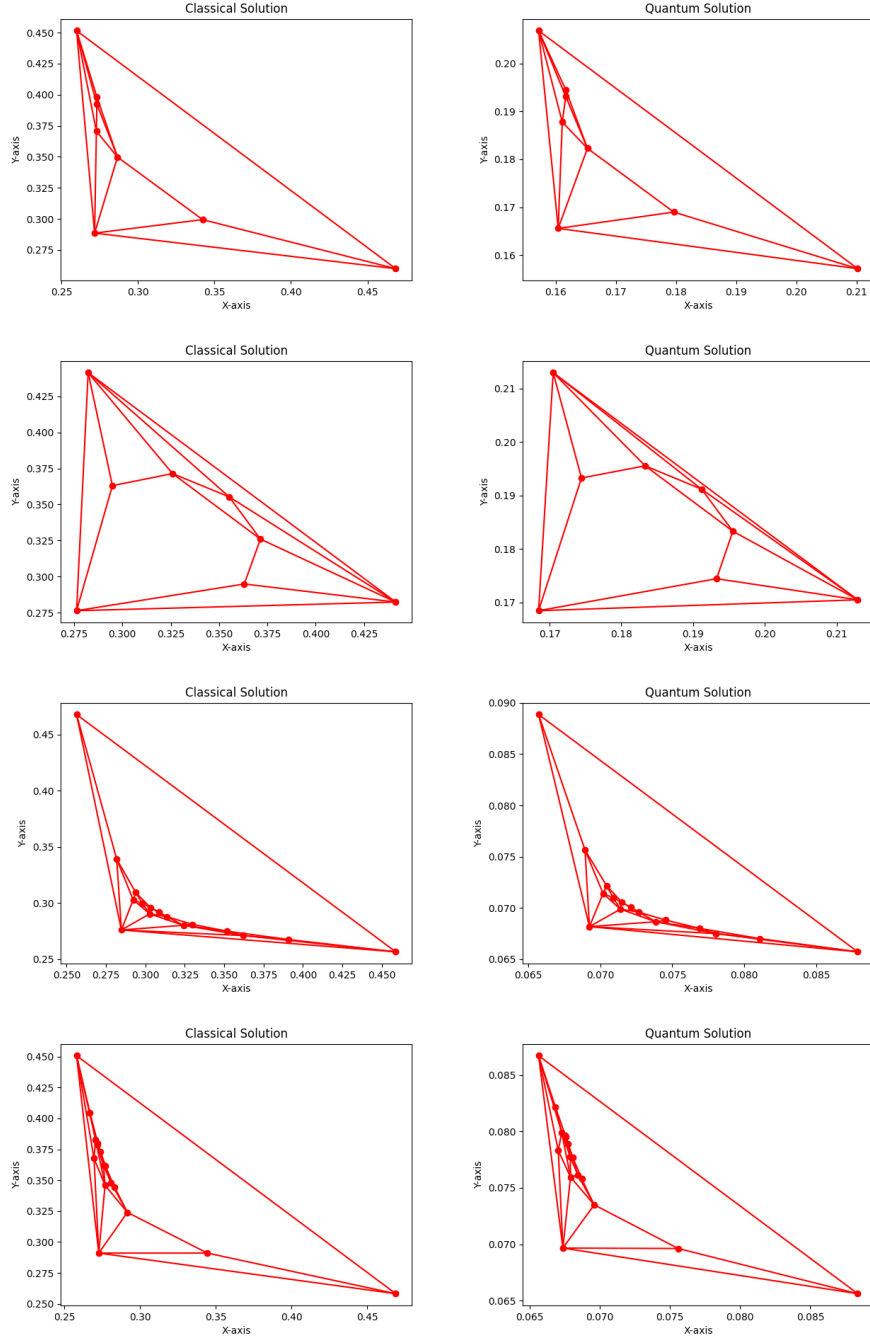


Fig. 5: Graph drawing results using both the classical and quantum solver. The first two are drawing random planar graphs with 8 nodes, and the last two are drawing random planar graphs with 16 nodes.

Furthermore, we use the circuit simulator and thus only demonstrate the results for small-size graphs. The time required for running the quantum simulator grows exponentially with the number of qubits, and even for a graph with 16 nodes, this required using matrices with size 2^{13} by 2^{13} . This makes simulating the quantum solver for larger-size graphs very difficult. Some of the resulting graphs are shown in Figures 4 and 5.

The graphs drawn were generated randomly by placing N random points on the plane, then defining their edges via a Delaunay triangulation. This guarantees their planarity as well as strong connectivity while maintaining low average degree per vertex. The dummy outer face is removed from the final drawing, leading to graphs whose outer faces are not perfectly placed triangles as their coordinates were not fixed.

Acknowledgements

We would like to thank David Eppstein for helpful discussions regarding the topics of this paper. This work was supported in part by NSF grant 2212129.

References

1. Bassoli, R., Boche, H., Deppe, C., Ferrara, R., Fitzek, F.H., Janssen, G., Saedinaeeni, S.: Quantum Communication Networks, vol. 23. Springer (2021)
2. Cabello, A., Danielsen, L.E., López-Tarrida, A.J., Portillo, J.R.: Quantum social networks. *Journal of Physics A: Mathematical and Theoretical* 45(28), 285101 (jun 2012), <https://dx.doi.org/10.1088/1751-8113/45/28/285101>
3. Childs, A.M., Kothari, R., Somma, R.D.: Quantum Algorithm for Systems of Linear Equations with Exponentially Improved Dependence on Precision. *SIAM Journal on Computing* 46(6), 1920–1950 (Jan 2017)
4. Chung, F.R.: *Spectral Graph Theory*, vol. 92. American Mathematical Soc. (1997)
5. Cross, A.: The IBM Q experience and QISKit open-source quantum computing software. In: APS March Meeting Abstracts. vol. 2018, pp. L58–003 (2018)
6. Dervovic, D., Herbster, M., Mountney, P., Severini, S., Usher, N., Wossnig, L.: Quantum linear systems algorithms: A primer (Feb 2018)
7. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall (1999)
8. Fáry, I.: On straight-line representation of planar graphs. *Acta Scientiarum Mathematicarum* 11(2), 229–233 (1948)
9. Floater, M.S.: Parametric tilings and scattered data approximation. *International Journal of Shape Modeling* 04(03n04), 165–182 (1998)
10. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.* 103, 150502 (Oct 2009), <https://link.aps.org/doi/10.1103/PhysRevLett.103.150502>
11. Hopcroft, J.E., Kahn, P.J.: A paradigm for robust geometric algorithms. *Algorithmica* 7(1-6), 339–380 (1992)
12. Kelner, J.A., Orecchia, L., Sidford, A., Zhu, Z.A.: A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In: 45th ACM Symposium on Theory of Computing (STOC). pp. 911–920 (2013)
13. Kitaev, A.Y.: Quantum measurements and the abelian stabilizer problem (1995)
14. Kobourov, S.G.: Force-directed drawing algorithms. In: Tamassia, R. (ed.) *Handbook of Graph Drawing and Visualization*, chap. 12, pp. 383–408. CRC Press (2013)
15. Kumar, R., Kumari, S., Bala, M.: Quantum mechanical model of information sharing in social networks. *Social Network Analysis and Mining* 11(1), 42 (2021)
16. Matta, C.F.: *Quantum Biochemistry*. John Wiley & Sons (2010)
17. Nielsen, M.A., Chuang, I.: *Quantum Computation and Quantum Information*. Cambridge University Press, 10th edn. (2010)
18. Pirani, M., Sundaram, S.: On the Smallest Eigenvalue of Grounded Laplacian Matrices. *IEEE Transactions on Automatic Control* pp. 1–1 (2015)
19. Spielman, D.A., Teng, S.H.: Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In: 36th ACM Symposium on Theory of Computing (STOC). pp. 81–90 (2004)
20. Stein, S.K.: Convex maps. *Proceedings of the American Mathematical Society* 2(3), 464–466 (1951)
21. Trotter, H.F.: On the product of semi-groups of operators. *Proceedings of the American Mathematical Society* 10(4), 545–551 (1959)
22. Tutte, W.T.: How to draw a graph. *Proceedings of the London Mathematical Society* 3(1), 743–767 (1963)
23. Wagner, K.: Bemerkungen zum vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung* 46, 26–32 (1936)

A A Brief Primer on Quantum Computing

Gate	Matrix Representation
I (Identity)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
X (Bitflip)	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
H	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
$P(\theta)$	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$
CX (Controlled Bitflip)	$\begin{bmatrix} I & \\ & X \end{bmatrix}$
Toffoli	$\begin{bmatrix} I & & \\ & I & \\ & & I \\ & & & X \end{bmatrix}$

Table 1: The matrix representation of quantum gates which will appear in this paper. The final two matrices are block encoded using the I and X single qubit gates. The controlled bitflip applies an X gate to the second qubit if the first qubit is set to 1. The Toffoli can be thought of as a doubly controlled bitflip, applying an X gate to a third qubit iff the first two qubits are set to 1.

In this appendix, we go over some notation and properties of quantum computing that are important in describing quantum algorithms. For a full introduction, we refer the reader to Nielsen and Chuang [17].

In a classical computer, an n -bit register can only store a single n -bit string at a time, but a quantum computer with an n -qubit register can store a **superposition** of n -bit strings. This is represented as a unit column vector referred to as a **state vector**, $|\psi\rangle = \sum_{i=0}^{n-1} \alpha_i |i\rangle$ (read "ket" psi), and the α_i 's are complex numbers such that their norm squared is equal to the probability of measuring the bit string i . Thus, the quantum states must always be a unit vector whose L_2 -norm $\sum_i |\alpha_i|^2$ equals 1. Quantum gates are represented as unitary matrices that act on state vectors, effectively altering the superposition at each application. Unitarity guarantees that the state vector remains a unit vector after the transformation.

An important operation in quantum computing is measurement. Although it is possible to store a superposition of states and operate on them concurrently, to read out any information about the state vector a measurement must be performed. The probability of measuring the bit string i is equal to $|\alpha_i|^2$, and

once measured, the state vector **collapses** to the bit string i , losing information about the state it collapsed from.³ Since the output of a measurement is drawn from a probability distribution determined by the squares of the amplitudes of the output state, quantum computation is inherently probabilistic in nature. The goal is to output a quantum state such that the “solution” bit string is measured with high probability. The probability of success can be boosted by repetition as with classical randomized algorithms.

B Example Source Code

We provide in Figure 6 example source code for our quantum simulations.

³ Although it is possible to perform different measurements, for simplicity we focus on measurement in the “standard basis” which measures the qubits as classical bit strings.

```

import numpy as np
import scipy
from linear_solvers import HHL
import random
from qiskit.quantum_info import Statevector

# Number of vertices
n = 8

# Generate a grounded Laplacian for input graph
L = Input graph Laplacian
A = L
A[0][0] += 1
A[1][1] += 1
A[2][2] += 1

b_x = np.array([0, 0, 1, 0, 0, 0, 0, 0])
b_y = np.array([0, 1, 0, 0, 0, 0, 0, 0])

def get_solution_vector(solution):
    """Extracts and normalizes simulated state vector
    from LinearSolverResult."""

    # Read out the relevant entries in the state vector
    half_dim = Statevector(solution.state).dim // 2
    solution_vector = Statevector(solution.state).data[half_dim:half_dim+n].real
    print(Statevector(solution.state).dim)

    return solution_vector

# Instantiate HHL object and solve for x and y coordinates.
hhl = HHL(epsilon=1e-4)
naive_hhl_solution_x = hhl.solve(A, b_x)
sol_x = get_solution_vector(naive_hhl_solution_x) * hhl.scaling
hhl = HHL(epsilon=1e-4)
naive_hhl_solution_y = hhl.solve(A, b_y)
sol_y = get_solution_vector(naive_hhl_solution_y) * hhl.scaling

```

Fig. 6: Example source code for our quantum simulation.