

Exponential algorithmic speedup by quantum walk

Andrew M. Childs,^{1,*} Richard Cleve,^{2,†} Enrico Deotto,^{1,‡}
Edward Farhi,^{1,§} Sam Gutmann,^{3,¶} and Daniel A. Spielman^{4,**}

¹*Center for Theoretical Physics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*

²*Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4*

³*Department of Mathematics, Northeastern University, Boston, MA 02115, USA*

⁴*Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*
(25 October 2002)

We construct an oracular (i.e., black box) problem that can be solved exponentially faster on a quantum computer than on a classical computer. The quantum algorithm is based on a continuous time quantum walk, and thus employs a different technique from previous quantum algorithms based on quantum Fourier transforms. We show how to implement the quantum walk efficiently in our oracular setting. We then show how this quantum walk can be used to solve our problem by rapidly traversing a graph. Finally, we prove that no classical algorithm can solve this problem with high probability in subexponential time.

I. INTRODUCTION

A primary goal of the field of quantum computation is to determine when quantum computers can solve problems faster than classical computers. Exponential quantum speedup has been demonstrated for a number of different problems, but in each case, the quantum algorithm for solving the problem relies on the quantum Fourier transform. The purpose of this paper is to demonstrate that exponential speedup can be achieved by a different algorithmic technique, the quantum walk. We show that quantum walks can solve an oracular computational problem exponentially faster than any classical algorithm.

Oracular problems provided the first examples of algorithmic speedup using a quantum instead of a classical computer. An oracular problem is specified in terms of a black box function, and the goal is to find some property of that function using as few queries to the black box as possible. In this setting, Deutsch gave an example of a problem that can be solved on a quantum computer using one query, but that requires two queries on a classical computer [1]. Deutsch and Jozsa generalized this problem to one that can be solved exactly on a quantum computer in polynomial time, but for which an exact solution on a classical computer requires exponential time [2]. However, this problem can be solved with high probability in polynomial time using a probabilistic classical algorithm. Bernstein and Vazirani gave the first example of a superpolynomial separation between probabilistic classical and quantum computation [3], and Simon gave another example in which the separation is exponential [4].

Quantum computers can also provide computational speedup over the best known classical algorithms for non-oracular problems. The first such example was provided by Shor, who gave polynomial-time algorithms for integer factorization and the discrete logarithm [5]. A number of generalizations and variations of these algorithms have been discovered for solving both oracular

* amchilds@mit.edu

† cleve@cpsc.ucalgary.ca

‡ deotto@mitlhs.mit.edu

§ farhi@mit.edu

¶ sgutman@neu.edu

** splehnan@math.mit.edu

and non-oracular problems with exponential speedup (e.g., [6, 7, 8, 9, 10, 11, 12, 13, 14, 15]). All of them are fundamentally based on quantum Fourier transforms.

We would like to find new techniques for achieving algorithmic speedup with quantum computers. Since many classical algorithms are based on random walks, it is natural to ask whether a quantum analogue of a random walk process might be useful for quantum computation. This idea was explored by Farhi and Gutmann [16], who proposed the model of a quantum walk used in the present paper.¹ They gave an example of a graph in which the time to propagate between two vertices (i.e., the hitting time) is exponentially faster for the quantum walk than for the corresponding classical random walk. A simpler example of a graph with an exponentially faster quantum hitting time was given in [17]. However, as we explain in Section II, these results do not imply algorithmic speedup. In the present paper, we modify the example presented in [17] to construct an oracular problem that can be solved efficiently using a quantum walk, but that no classical algorithm can solve in subexponential time. Although the quantum walk is defined as a continuous time quantum process, we show how to implement it in the conventional quantum computing paradigm with the structure of the graph provided in the form of an oracle. In other words, the algorithm consists of a polynomial-length sequence of applications of the oracle and unitary gates that act on a few qubits at a time.

We note that the quantum analogue of a classical random walk is not unique, and other models have been proposed. Whereas the states of the continuous time quantum walk used in this paper lie in a Hilbert space spanned by states corresponding to the vertices in the graph, these alternative models operate in discrete time and make use of an extra state space (e.g., to implement a “quantum coin”) [18, 19, 20, 21, 22]. As far as we know, our results are the first example of algorithmic speedup based on either discrete or continuous time quantum walks.

The structure of this paper is as follows. In Section II, we construct a family of graphs and use it to define our problem. In Section III, we present a quantum algorithm for solving this problem in polynomial time with high probability, and in Section IV, we show that the problem cannot be solved classically in subexponential time with high probability. We conclude with a discussion of the results in Section V.

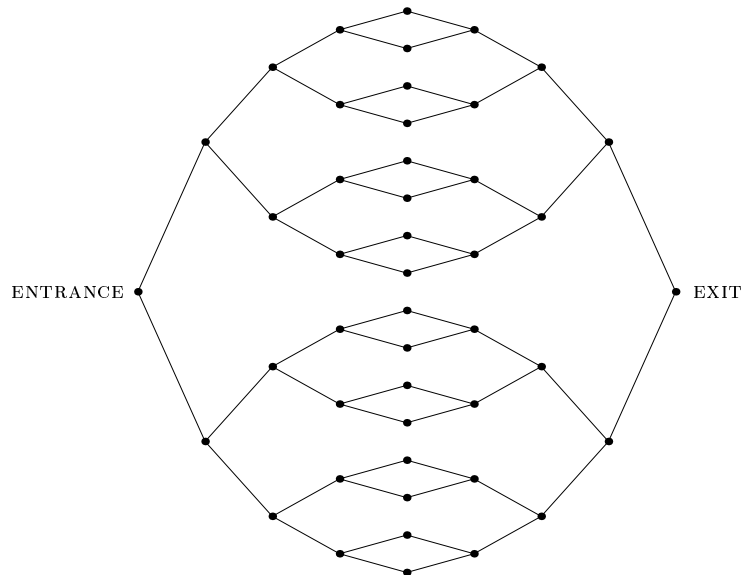
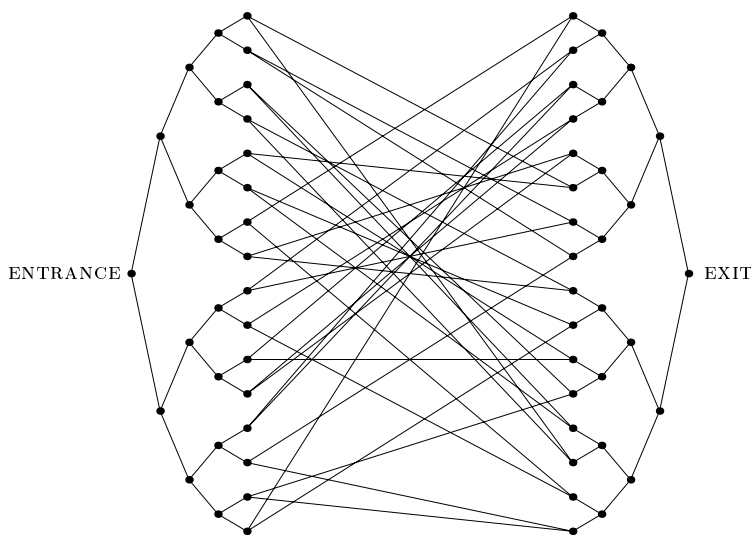
II. THE PROBLEM

In this section we describe in detail the problem we address in this paper. The problem involves determining a certain property of a graph whose structure is provided in the form of an oracle. A graph G is a set of N vertices and a set of edges that specify which pairs of vertices are connected.

Our problem is based on a generalization of the graphs presented in [17]. In that paper, the authors consider a sequence of graphs G_n consisting of two balanced binary trees of height n with the 2^n leaves of the left tree identified with the 2^n leaves of the right tree in the simple way shown in Figure 1 (for $n = 4$). A classical random walk starting at the left root of this graph requires exponentially many steps (in n) to reach the right root. However, the continuous time quantum walk traverses G_n in a time linear in n .

In this paper, we modify the graphs in the previous example so that the quantum walk is exponentially better not only than the corresponding classical random walk, but also than *any* classical algorithm one can design to traverse the graph. Before we describe the modified graphs, we provide an oracular (black box) setting in which graphs can be specified, and where our notion of an algorithm traversing a graph can be made precise. The vertices are named with randomly

¹ The term *quantum random walk* is somewhat misleading since the dynamics of this process are coherent, so we use the term *quantum walk* in this paper.

FIG. 1: The graph G_4 .FIG. 2: A typical graph G'_4 .

chosen $2n$ -bit strings, and the oracle only refers to the vertices by their names. Because G_n contains $O(2^n)$ vertices, $n + O(1)$ bits would be sufficient to give each vertex a unique name, but we use $2n$ bits for each name so that there are exponentially more possible names than vertices in the graph. The oracle takes a $2n$ -bit string as input, and if that name corresponds to an actual vertex in the graph, the oracle outputs the names of the adjacent vertices. There are also two identifiable vertices called ENTRANCE and EXIT, which in the case of G_n are the left root and the right root of the trees, respectively. The traversal problem is:

- Given an oracle for the graph and the name of the ENTRANCE, find the name of the EXIT.

However, even with G_n given by such an oracle, the EXIT can be found in polynomial time using a classical algorithm that is *not* a random walk. The key is that we can always tell whether a particular vertex is in the central column by checking its degree. We begin at the ENTRANCE.

At each vertex, we query the oracle and move to one of the two unvisited adjacent vertices. After n steps, we reach the central column and then proceed moving to unvisited adjacent vertices in the right tree, checking the degree at each step. If we discover that after k steps we are again at a central vertex, we know that the wrong move happened after $k/2$ steps (k can only be even due to the structure of G_n) and we can backtrack to that point and take the other edge. After only $O(n^2)$ steps this procedure will reach the EXIT.²

We now describe how the graphs G_n are modified so that they cannot be traversed efficiently with a classical algorithm. We choose a graph G'_n at random from a particular distribution on graphs. A typical graph G'_n is shown in Figure 2 (for $n = 4$). The distribution is defined as follows. The graph again consists of two balanced binary trees of height n , but instead of identifying the leaves, they are connected by a random cycle that alternates between the leaves of the two trees. In other words, we choose a leaf on the left at random and connect it to a leaf on the right chosen at random. Then we connect the latter to a leaf on the left chosen randomly among the remaining ones. We continue with this procedure, alternating sides, until every leaf on the left is connected to two leaves on the right (and vice versa). As we will show in Section IV, no subexponential time classical algorithm can with non-negligible probability find the EXIT given the ENTRANCE and a typical oracle for G'_n .

We will show in Sections III C and III D that the quantum walk goes from ENTRANCE to EXIT in any graph G'_n in polynomial time. However, to construct a quantum algorithm, we must show how to *implement* the quantum walk, i.e., how to efficiently simulate the appropriate Hamiltonian evolution on a quantum computer. We do this in the conventional paradigm for oracular quantum computation, and our implementation will work for a general graph. The algorithm will consist of a sequence of unitary operators that are either the oracle or act on a few qubits at a time. We address the general implementation issue in Section III B. In order to implement the quantum walk on a general graph, we must also be given a consistent coloring of the edges of the graph. In other words, there must be $k = \text{poly}(\log(\#\text{vertices}))$ colors assigned to the edges of the graph such that all the edges incident on a given vertex are different colors. The input to the oracle consists of both the name of a vertex and a particular color, and the output name will depend on the input color, as explained in detail in Section III B.

In principle, knowledge of such a coloring might help a classical algorithm solve our problem, because there could be correlations between the colors of the edges in different places in the graph. However, we will show in Section IV that there exists a consistent (random) coloring of any graph G'_n , using only nine colors (independent of n), that is completely useless to any classical algorithm. Consequently, this modification of the oracle can be made irrelevant from the classical point of view. For an alternative implementation of the quantum walk on the graphs G'_n that does not require colors, see Appendix B.

III. QUANTUM ALGORITHM

In this section, we discuss quantum walks in general and then use them to present a quantum algorithm for solving our problem. We begin by describing the model of a continuous time quantum walk on any graph in Section III A. We define the quantum walk to be Schrödinger evolution with

² As an aside, we note that there is also a polynomial-time classical traversal algorithm for the n -dimensional hypercube (where one vertex is designated as the ENTRANCE and the EXIT is the unique vertex whose distance from ENTRANCE is n). For a description of the algorithm, see Appendix A. This means that there can be no exponential algorithmic speedup using quantum walks to traverse the hypercube, even though both continuous and discrete time quantum walks have been shown to reach the EXIT in a polynomial number of steps in a non-oracular setting [23, 24].

the Hamiltonian equal to (a constant multiple of) the adjacency matrix of the graph. We explain how to implement the quantum walk on a general graph in an oracular setting on a quantum computer in Section III B. In Section III C we show that the quantum walk on the graph G'_n starting at the ENTRANCE can be viewed as propagation on a discrete line, and we give a sequence of arguments explaining why the walk propagates through the graph in linear time. Finally, in Section III D, we prove that the quantum walk reaches the EXIT using a polynomial number of calls to the oracle.

A. Quantum walk

We now formally describe our model of a quantum walk on a general graph G . We write $a \in G$ to denote that the vertex a is in the graph and $aa' \in G$ to denote that the edge joining vertices a and a' is in the graph. We let $d(a)$ denote the degree of vertex a , i.e., the number of edges incident on that vertex.

Our model of a quantum walk is defined in close analogy to a continuous time classical random walk, which is a Markov process. In a continuous time classical random walk, there is a fixed probability per unit time γ of moving to an adjacent vertex. In other words, from any vertex, the probability of jumping to any connected vertex in a time ϵ is $\gamma\epsilon$ (in the limit $\epsilon \rightarrow 0$). If the graph has N vertices, this classical random walk can be described by the $N \times N$ infinitesimal generator matrix K defined by

$$K_{aa'} = \begin{cases} \gamma & a \neq a', aa' \in G \\ 0 & a \neq a', aa' \notin G \\ -d(a)\gamma & a = a'. \end{cases} \quad (1)$$

If $p_a(t)$ is the probability of being at vertex a at time t , then

$$\frac{dp_a(t)}{dt} = \sum_{a'} K_{aa'} p_{a'}(t). \quad (2)$$

Note that because the columns of K sum to zero, an initially normalized distribution remains normalized, i.e., $\sum_a p_a(t) = 1$ for all t .

Now consider quantum evolution in an N -dimensional Hilbert space spanned by states $|1\rangle, |2\rangle, \dots, |N\rangle$ corresponding to the vertices of G . If the Hamiltonian is H , then the dynamics of the system are determined by the Schrödinger equation,

$$i \frac{d}{dt} \langle a | \psi(t) \rangle = \sum_{a'} \langle a | H | a' \rangle \langle a' | \psi(t) \rangle. \quad (3)$$

Note the similarity between (2) and (3). A natural quantum analogue to the continuous time classical random walk described above is given by the Hamiltonian with matrix elements [16]

$$\langle a | H | a' \rangle = K_{aa'}. \quad (4)$$

In the quantum case, there is no need for the columns of H to sum to zero; we only require H to be Hermitian. We set the diagonal to zero for simplicity. In this paper, we define a quantum walk in terms of a Hamiltonian with matrix elements

$$\langle a | H | a' \rangle = \begin{cases} \gamma & a \neq a', aa' \in G \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

In other words, H is the adjacency matrix of the graph times γ . Because H is Hermitian, probability is conserved, i.e., $\sum_a |\langle a | \psi(t) \rangle|^2 = 1$ for all t .

B. Implementing the quantum walk

In this section, we describe how to implement the quantum walk on a general graph G using a universal quantum computer. Our goal is to simulate the unitary evolution e^{-iHt} with H given by (5). We want to do this with the graph given to us in the form of an oracle. However, in the oracular setting, we have only been able to implement the quantum walk on a general graph using additional structure: we require that the graph comes with a consistent coloring. We assume that each edge of the graph is assigned a color, where the total number of colors is $k = \text{poly}(\log N)$. We say that a coloring is consistent if no vertex is incident with two edges of the same color. Although we might want to use more than the minimal number of colors for a particular application, any graph whose maximum degree is Δ can be consistently colored with at most $\Delta + 1$ colors [25]. For the *specific* problem addressed in this paper, we will show that the oracle can provide a consistent coloring that cannot be used by any classical algorithm to help solve the problem, since a classical algorithm could make up the coloring as it goes. Alternatively, we could implement the quantum walk for this particular problem using a similar idea in which the coloring is generated by the quantum algorithm. We outline this construction in Appendix B.

We now give our method for implementing the quantum walk on a general graph G for which the oracle has assumed a particular consistent coloring of the graph. Let $n = \lceil \log N \rceil$ so it takes n bits to list the vertices of the graph G . In the classical setting, the black box takes two inputs, a name a given as a $2n$ -bit string and a color c . If the input name a corresponds to a vertex that is incident with an edge of color c , then the output is the name of the vertex joined by that edge. If a is not the name of a vertex or if a is the name of a vertex but there is no incident edge of color c , then the output is the special bit string $11\dots 1$, which is not the name of any vertex. For shorthand, we write $v_c(a) = a'$, where a is the input name, c is the color, and a' is the output name. If $a' \neq 11\dots 1$, then a' is the name of the vertex that is connected by an edge of color c to the vertex named a . Note that $v_c(v_c(a)) = a$ for $v_c(a) \neq 11\dots 1$, which is a crucial ingredient in our implementation of the quantum walk.

The unitary quantum black box corresponding to this classical black box is described as follows. Let a, b be $2n$ -bit strings and let c be a color. Then the action of the quantum black box U associated with the graph G is

$$U|a, b, c\rangle = |a, b \oplus v_c(a), c\rangle \quad (6)$$

where \oplus denotes bitwise addition modulo 2. However, we will never need to query U with a superposition of different colors, so for our purposes, it is sufficient to omit the color register and assume that we have access to the k unitary transformations

$$U_c|a, b\rangle = |a, b \oplus v_c(a)\rangle, \quad (7)$$

one for each of the k possible colors. Clearly, this operation can be performed using the oracle (6). In fact, by checking whether $v_c(a) = 11\dots 1$, it is also straightforward to extend the Hilbert space by a single qubit and perform each of the k transformations

$$V_c|a, b, r\rangle = |a, b \oplus v_c(a), r \oplus f_c(a)\rangle, \quad (8)$$

where

$$f_c(a) = \begin{cases} 0 & v_c(a) \neq 11\dots 1 \\ 1 & v_c(a) = 11\dots 1, \end{cases} \quad (9)$$

using only a few calls to (6).³ In summary, for our purposes, we may view the oracle as giving us the ability to perform each of the k unitary transformations V_c given by (8), where a, b are $2n$ -bit strings and r is a single bit.

The Hilbert space that V_c acts on in (8) has dimension 2^{4n+1} , which is much larger than the number of vertices in the graph. We want the quantum evolution e^{-iHt} to take place in an N -dimensional subspace. To this end, we will show how to implement the quantum walk given the oracles V_c in the sense that we will construct a Hamiltonian H satisfying

$$H|a, 0, 0\rangle = \sum_{c: v_c(a) \in G} |v_c(a), 0, 0\rangle. \quad (10)$$

This means that if we start our evolution in the subspace of states of the form $|a, 0, 0\rangle$ where $a \in G$, then we will remain in this N -dimensional subspace.

In a non-oracular setting, we say we can efficiently simulate any m -qubit Hamiltonian H if we can approximate the unitary evolution e^{-iHt} for any $t = \text{poly}(m)$ using a polynomial number of one- and two-qubit unitary gates. In our oracular setting, we regard a simulation of H as efficient if it can approximate e^{-iHt} for any $t = \text{poly}(m)$ using a polynomial number of calls to the oracle and a polynomial number of additional one- and two-qubit gates. In general, in either setting, it is not easy to answer the question of whether a particular Hamiltonian can be simulated efficiently.⁴ However, there are some useful standard tools for simulating Hamiltonians. The following is a list of five such tools, four of which will be used in our construction.

1. *Local terms.* If H acts on $O(1)$ qubits, it can be simulated. This is simply because any operator acting on a constant number of qubits can be approximated using a constant number of one- and two-qubit gates.
2. *Linear combination.* If we can simulate H_1, \dots, H_k , then we can simulate $H_1 + \dots + H_k$ as a result of the Lie product formula

$$e^{-i(H_1 + \dots + H_k)t} = \left(e^{-iH_1 t/j} \dots e^{-iH_k t/j} \right)^j + O(k \| [H_p, H_q] \| t^2 / j), \quad (11)$$

where $\| [H_p, H_q] \|$ is the largest norm of a commutator of two of the Hamiltonians. This is a powerful tool for the simulation of *physical* quantum systems, which can typically be expressed as a sum of local terms [26].

3. *Commutation.* If we can simulate H_1 and H_2 , then we can simulate $i[H_1, H_2]$. This is a consequence of the identity

$$e^{[H_1, H_2]t} = \lim_{j \rightarrow \infty} \left(e^{-iH_1 t/\sqrt{j}} e^{-iH_2 t/\sqrt{j}} e^{iH_1 t/\sqrt{j}} e^{iH_2 t/\sqrt{j}} \right)^j. \quad (12)$$

Using linear combination and commutation, it is possible to simulate any Hamiltonian in the Lie algebra generated by a set of Hamiltonians. We will not need to use commutation in the present paper, but we include it for completeness.

4. *Unitary conjugation.* If we can simulate H and we can efficiently perform the unitary operation U , then we can simulate UHU^\dagger . This follows from the simple fact that $Ue^{-iHt}U^\dagger = e^{-iUHU^\dagger t}$.

³ Note that in (8), the register containing r consists of a single qubit, as opposed to (6), in which c represents one of k colors.

⁴ Certainly *most* Hamiltonians cannot be simulated efficiently, since there are many more possible m -qubit Hamiltonians than strings of $\text{poly}(m)$ one- and two-qubit gates.

5. *Tensor product.* If H_j each act on $O(1)$ qubits and the product of their eigenvalues is efficiently computable, then we can simulate $\bigotimes_j H_j$. This can be done with a generalization of a technique discussed in Section 4.7.3 of [27]. We will not present the general construction, but we show its application to a specific Hamiltonian below. Note that $\bigotimes_j H_j$ may be highly nonlocal, so it may look very different from the Hamiltonian of a physical system.

We now use these tools to construct the quantum walk on a graph G specified by the oracles V_c given by (8). In other words, we show how to efficiently simulate (10). The Hilbert space will consist of states of the form $|a, b, r\rangle$, where a and b are $2n$ -bit strings and r is a single bit. The states that correspond to vertices are $|a, 0, 0\rangle$, where a is the name of a vertex in the graph.

We begin by showing how to simulate the evolution generated by the Hermitian operator T satisfying

$$T|a, b, 0\rangle = |b, a, 0\rangle \quad (13)$$

$$T|a, b, 1\rangle = 0. \quad (14)$$

We will use T as a building block in our simulation of H . Our simulation of T uses only $O(n)$ one- and two-qubit gates.

The operator T may be written as

$$T = \left(\bigotimes_{l=1}^{2n} S^{(l, 2n+l)} \right) \otimes |0\rangle\langle 0| \quad (15)$$

where the superscript indicates which two qubits S acts on, and the projector onto $|0\rangle$ acts on the third register. Here S is a Hermitian operator on two qubits satisfying $S|z_1 z_2\rangle = |z_2 z_1\rangle$. Since the eigenvalues of S are ± 1 , the eigenvalues of T are $0, \pm 1$, and they are easy to compute. Thus e^{-iTt} can be simulated with the circuit shown in Figure 3. In this figure, W denotes a two-qubit unitary operator that diagonalizes S . The unique eigenvector of S with eigenvalue -1 is $\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$, so we take

$$W|00\rangle = |00\rangle \quad (16)$$

$$W \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) = |01\rangle \quad (17)$$

$$W \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) = |10\rangle \quad (18)$$

$$W|11\rangle = |11\rangle. \quad (19)$$

Applying $W^{\otimes 2n}$ diagonalizes T , and the Toffoli gates compute the argument of the eigenvalue in an ancilla register initially prepared in the state $|0\rangle$. Note that a filled circle denotes “control on $|1\rangle$ ” and an open circle denotes “control on $|0\rangle$.” After computing the eigenvalue, we apply the appropriate phase shift if $r = 0$ by evolving for a time t according to the Pauli Z operator satisfying $Z|z\rangle = (-1)^z|z\rangle$. The controlled phase shift can be performed since it is a local term. Finally, we uncompute the eigenvalue and return to the original basis.

Our Hamiltonian for the quantum walk on G is

$$H = \sum_c V_c^\dagger T V_c \quad (20)$$

where the sum runs over all k colors. Given the simulation of T described above, this can be simulated using unitary conjugation and linear combination. Note that $V_c^\dagger = V_c$, so we do not need to separately explain how to perform V_c^\dagger .

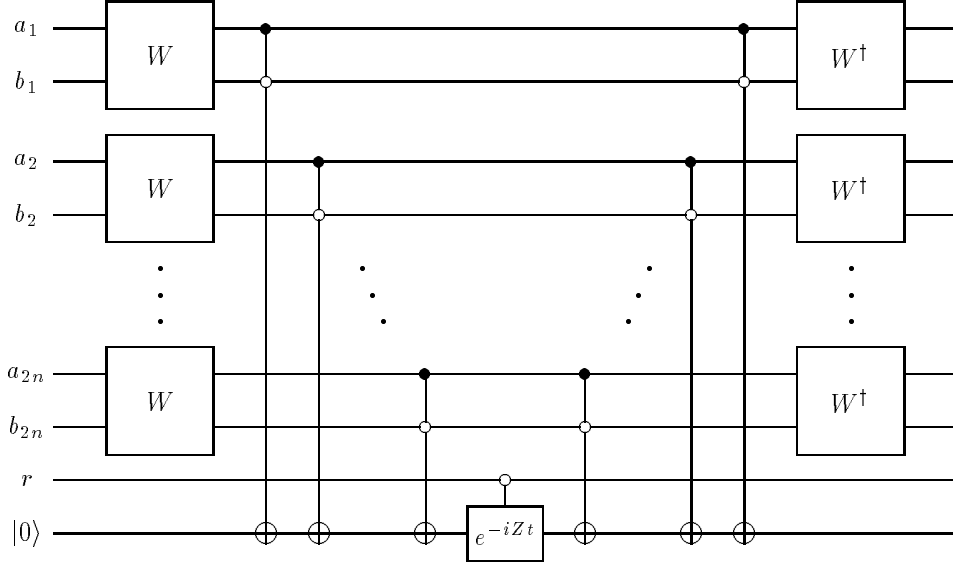


FIG. 3: A circuit for simulating e^{-iZt} .

To conclude, we show that (20) acts as it should to satisfy (10). We have

$$H|a, 0, 0\rangle = \sum_c V_c T|a, v_c(a), f_c(a)\rangle \quad (21)$$

$$= \sum_c \delta_{0, f_c(a)} V_c |v_c(a), a, 0\rangle \quad (22)$$

$$= \sum_{c: v_c(a) \in G} |v_c(a), a \oplus v_c(v_c(a)), f_c(v_c(a))\rangle. \quad (23)$$

Noting that $v_c(a) \in G \Rightarrow f_c(v_c(a)) = 0$ and $v_c(v_c(a)) = a$, this shows that

$$H|a, 0, 0\rangle = \sum_{c: v_c(a) \in G} |v_c(a), 0, 0\rangle \quad (24)$$

as desired.

Although the continuous time quantum walk is most naturally viewed as Schrödinger evolution according to a time-independent Hamiltonian, we have shown how to simulate this evolution in the conventional circuit model of quantum computation. Note that since the names of the vertices may be arbitrary, the graph Hamiltonian may not be local, but nevertheless it can be simulated in the circuit model.

C. Propagation on a line

We now consider the quantum walk on the specific family of graphs G'_n of Figure 2. In this section, we use standard physics techniques to give compelling evidence that the quantum walk propagates from the ENTRANCE to the EXIT in linear time. We explain how the walk on G'_n can be viewed as a walk on a finite line with a defect at the center, and we show through a series of examples that the defect and the boundaries do not significantly affect the walk. In section III D, we prove that the walk reaches the EXIT in polynomial time.

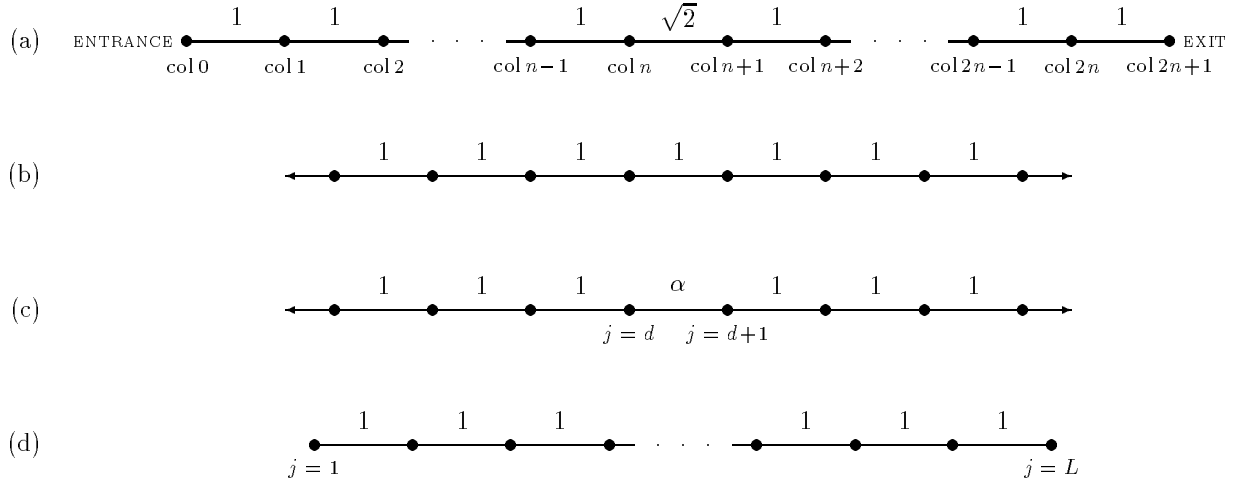


FIG. 4: Quantum walks on lines. (a) Reduction of the quantum walk on G'_n to a quantum walk on a line. (b) Quantum walk on an infinite, translationally invariant line. (c) Quantum walk on an infinite line with a defect. (d) Quantum walk on a finite line without a defect.

The analysis of the walk on G'_n is particularly simple because it can be reduced to a walk on a line with $2n + 2$ vertices, one for each column of the original graph. Consider the $(2n + 2)$ -dimensional subspace spanned by the states

$$|\text{col } j\rangle = \frac{1}{\sqrt{N_j}} \sum_{a \in \text{column } j} |a\rangle, \quad (25)$$

where

$$N_j = \begin{cases} 2^j & 0 \leq j \leq n \\ 2^{2n+1-j} & n + 1 \leq j \leq 2n + 1. \end{cases} \quad (26)$$

We refer to this subspace as the *column subspace*. Because every vertex in column j is connected to the same number of vertices in column $j + 1$ and every vertex in column $j + 1$ is connected to the same number of vertices in column j , applying H to any state in the column subspace results in another state in this subspace. Despite the random connections in G'_n , the column subspace is invariant under H . In particular, a quantum walk starting in the state corresponding to the ENTRANCE always remains in the column subspace. Thus, to understand the quantum walk starting from the ENTRANCE, we only need to understand how the Hamiltonian acts on the column subspace. In this subspace, the non-zero matrix elements of H are

$$\langle \text{col } j | H | \text{col}(j + 1) \rangle = \begin{cases} \sqrt{2}\gamma & 0 \leq j \leq n - 1, \quad n + 1 \leq j \leq 2n \\ 2\gamma & j = n \end{cases} \quad (27)$$

(and those deduced by Hermiticity of H). For simplicity, we set $\gamma = 1/\sqrt{2}$. The quantum walk in the column subspace is shown pictorially in Figure 4(a).

We claim that if the quantum state at $t = 0$ is $|\text{col } 0\rangle$, then at a time of order $n/2$, there is an appreciable probability of being at $|\text{col}(2n + 1)\rangle$. To see this, first consider propagation on an infinite, translationally invariant line as shown in Figure 4(b). The nonzero matrix elements of the Hamiltonian are

$$\langle j | H | j \pm 1 \rangle = 1, \quad -\infty < j < \infty. \quad (28)$$

The eigenstates of this Hamiltonian are the momentum eigenstates $|p\rangle$ with components

$$\langle j|p\rangle = \frac{1}{\sqrt{2\pi}} e^{ipj}, \quad -\pi \leq p \leq \pi \quad (29)$$

having energies

$$E_p = 2 \cos p. \quad (30)$$

The propagator, or Green's function, to go from j to k in time t is

$$G(j, k, t) = \langle k|e^{-iHt}|j\rangle \quad (31)$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} dp e^{ip(k-j)-2it \cos p} \quad (32)$$

$$= (-i)^{k-j} J_{k-j}(2t), \quad -\infty < j, k < \infty \quad (33)$$

where $J_\nu(\cdot)$ is a Bessel function of order ν . By the well-known properties of the Bessel function, this shows that a state initially localized in a single column evolves as a left moving and a right moving wave packet, each propagating with speed 2. To see this, note that the Bessel function has the following asymptotic expansions for $\nu \gg 1$:

$$J_\nu(\nu \operatorname{sech} \xi) \sim \frac{e^{-\nu(\xi - \tanh \xi)}}{\sqrt{2\pi\nu \tanh \xi}} \quad (34)$$

$$J_\nu(\nu + \xi\nu^{1/3}) = (2/\nu)^{1/3} \operatorname{Ai}(-2^{1/3}\xi) + O(\nu^{-1}) \quad (35)$$

$$J_\nu(\nu \sec \xi) = \sqrt{\frac{2}{\pi\nu \tan \xi}} \left\{ \cos\left[\frac{\pi}{4} - \nu(\xi - \tan \xi)\right] + O(\nu^{-1}) \right\}, \quad 0 < \xi < \frac{\pi}{2} \quad (36)$$

where $\operatorname{Ai}(\cdot)$ is an Airy function [28]. These three relations show that for $|k-j| \gg 1$, $G(j, k, t)$ is exponentially small in $|k-j|$ for $t < 0.99 \cdot |k-j|/2$, of order $|k-j|^{-1/3}$ for t near $|k-j|/2$, and of order $|k-j|^{-1/2}$ for $t > 1.01 \cdot |k-j|/2$.

To understand the effect of a defect, we now consider an infinite line with a defect between sites $j = d$ and $j = d + 1$, as shown in 4(c). For generality, we consider the case where the matrix element across the defect is α . We use standard scattering theory to calculate the transmission coefficient for an incident plane wave of momentum $p > 0$. Consider a state $|\psi\rangle$ consisting of an incident plane wave and a reflected plane wave on the left side of the defect and a transmitted plane wave on the right side:

$$\langle j|\psi\rangle = \begin{cases} \frac{1}{\sqrt{2\pi}} e^{ipj} + \frac{\mathcal{R}}{\sqrt{2\pi}} e^{-ipj} & j \leq d \\ \frac{\mathcal{T}}{\sqrt{2\pi}} e^{ipj} & j \geq d + 1. \end{cases} \quad (37)$$

Here \mathcal{R} is the reflection coefficient and \mathcal{T} is the transmission coefficient. Requiring this to be an eigenstate of the Hamiltonian, we find

$$\mathcal{T}(p) = \frac{2i\alpha \sin p}{(\alpha^2 - 1) \cos p + i(\alpha^2 + 1) \sin p} \quad (38)$$

which gives

$$|\mathcal{T}(p)|^2 = \frac{8 \sin^2 p}{1 + 8 \sin^2 p} \quad (39)$$

for $\alpha = \sqrt{2}$, as shown in Figure 5. A narrow wave packet with momentum p propagates through the defect with probability $|\mathcal{T}(p)|^2$. The wave packet that results from a state initially localized at

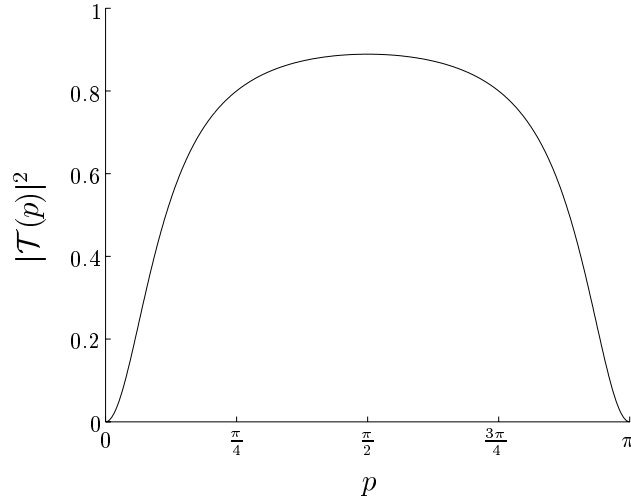


FIG. 5: The transmission probability $|\mathcal{T}(p)|^2$ as a function of momentum for the infinite line with a defect (with $\alpha = \sqrt{2}$).

a particular site is spread out over a range of momenta, but since the transmission probability is appreciable over most of the range, it is clear that there will be substantial transmission through the defect.

Finally, to show that boundaries do not impede the propagation, we now consider the finite line (without a defect) shown in Figure 4(d). This problem can be treated by standard techniques of multiple reflection. The exact Green's function $\tilde{G}(j, k, t)$ for this finite line in terms of the Green's function $G(j, k, t)$ for the infinite line is⁵

$$\tilde{G}(j, k, t) = \sum_{l=-\infty}^{\infty} [G(j, k + 2l(L + 1), t) - G(j, -k + 2l(L + 1), t)], \quad 1 \leq j, k \leq L. \quad (40)$$

This can be interpreted as a sum of paths making reflections off the boundaries. To verify this formula, the reader can check that it satisfies the Schrödinger equation, the boundary conditions, and the initial condition. The Schrödinger equation is satisfied because each term individually satisfies it for the infinite line. The boundary conditions $\tilde{G}(j, 0, t) = \tilde{G}(j, L + 1, t) = 0$ can be verified by substitution. The initial condition $\tilde{G}(j, k, 0) = \delta_{jk}$ holds because $G(j, k, 0) = \delta_{jk}$, and the only contribution at $t = 0$ comes from the first term of (40) with $l = 0$.

We will now see, using the particular form of the propagator in terms of a Bessel function, that propagation from $j = 1$ to $j = L$ takes time $L/2$ for $L \gg 1$. Since $J_{k-j}(2t)$ is small for $|k - j| \gg 2t$, there are only four terms that contribute significantly for $t \gtrsim L/2$. They result from taking $l = 0, -1$ in the first term of (40) and $l = 0, 1$ in the second term. The resulting expression is

$$\tilde{G}(1, L, t) \approx G(1, L, t) + G(1, -L - 2, t) - G(1, -L, t) - G(1, L + 2, t), \quad t \gtrsim L/2, \quad (41)$$

the magnitude of which is not small.

We have seen that propagation on a line occurs as a wave packet with speed 2, that the wave packet is substantially transmitted through a defect, and that reflections off boundaries do not

⁵ Note that (40) is the exact formula for propagation in the graph G_n shown in Figure 1 and studied in [17] if we use (5) instead of (4) as the definition of a quantum walk.

impede propagation. Taken together, these facts constitute compelling evidence that the quantum walk traverses the graph G'_n in linear time. The exact propagator for the line shown in Figure 4(a) can be calculated using a more sophisticated version of these techniques [29]. This exact propagator, evaluated for t near n , is of order $n^{-1/3}$. We spare the reader the details of this calculation and give a simpler proof of a bound that is not tight—but is nevertheless polynomial—in the following section.

D. Upper bound on the hitting time

Although the preceding section demonstrated beyond any reasonable doubt that the quantum walk traverses the graph G'_n in linear time, we now provide a simple proof that the hitting time is upper bounded by a polynomial.

For the purpose of this proof, it will be more convenient to consider the graph G'_{n-1} , which reduces to a line with $2n$ vertices. We label the vertices from 1 to $2n$, and the defect is on the edge between vertices n and $n+1$. With this labeling and $\gamma = 1/\sqrt{2}$, the Hamiltonian (27) is

$$\langle \text{col } j | H | \text{col}(j+1) \rangle = \begin{cases} 1 & 1 \leq j \leq n-1, \quad n+1 \leq j \leq 2n-1 \\ \sqrt{2} & j = n, \end{cases} \quad (42)$$

with Hermiticity of H giving the other nonzero matrix elements.

Define a reflection operator

$$R | \text{col } j \rangle = | \text{col}(2n+1-j) \rangle. \quad (43)$$

Note that $R^2 = 1$, so R has eigenvalues ± 1 . R commutes with H on the column subspace, so we can find simultaneous eigenstates of R and H . These are of the form

$$\langle \text{col } j | E \rangle = \begin{cases} \sin pj & 1 \leq j \leq n \\ \pm \sin(p(2n+1-j)) & n+1 \leq j \leq 2n, \end{cases} \quad (44)$$

which explicitly vanish at $j = 0$ and $j = 2n+1$. The eigenvalue corresponding to the eigenstate $|E\rangle$ is $E = 2 \cos p$, and the quantization condition (to be discussed later) comes from matching at vertices n and $n+1$. The ENTRANCE vertex corresponds to $| \text{col } 1 \rangle$ and the EXIT vertex to $| \text{col } 2n \rangle$.

Lemma 1 *Consider the quantum walk in G'_{n-1} starting at the ENTRANCE. Let the walk run for a time t chosen uniformly in $[0, \tau]$ and then measure in the computational basis. If $\tau \geq \frac{4n}{\epsilon \Delta E}$ for any constant $\epsilon > 0$, where ΔE is the magnitude of the smallest gap between any pair of eigenvalues of the Hamiltonian, then the probability of finding the EXIT is greater than $\frac{1}{2n}(1 - \epsilon)$.*

Proof The probability of finding the EXIT after the randomly chosen time $t \in [0, \tau]$ is

$$\begin{aligned} & \frac{1}{\tau} \int_0^\tau dt | \langle \text{col } 2n | e^{-iHt} | \text{col } 1 \rangle |^2 \\ &= \frac{1}{\tau} \sum_{E, E'} \int_0^\tau dt e^{-i(E-E')t} \langle \text{col } 2n | E \rangle \langle E | \text{col } 1 \rangle \langle \text{col } 1 | E' \rangle \langle E' | \text{col } 2n \rangle \end{aligned} \quad (45)$$

$$\begin{aligned} &= \sum_E | \langle E | \text{col } 1 \rangle |^2 | \langle E | \text{col } 2n \rangle |^2 \\ &+ \sum_{E \neq E'} \frac{1 - e^{-i(E-E')\tau}}{i(E-E')\tau} \langle \text{col } 2n | E \rangle \langle E | \text{col } 1 \rangle \langle \text{col } 1 | E' \rangle \langle E' | \text{col } 2n \rangle. \end{aligned} \quad (46)$$

Because of (44), we have $\langle E | \text{col } 1 \rangle = \pm \langle E | \text{col } 2n \rangle$. Thus the first term is

$$\sum_E |\langle E | \text{col } 1 \rangle|^4 \geq \frac{1}{2n} \quad (47)$$

as is easily established using the Cauchy-Schwartz inequality. The second term can be bounded as follows:

$$\begin{aligned} & \left| \sum_{E \neq E'} \frac{1 - e^{-i(E-E')\tau}}{i(E-E')\tau} \langle \text{col } 2n | E \rangle \langle E | \text{col } 1 \rangle \langle \text{col } 1 | E' \rangle \langle E' | \text{col } 2n \rangle \right| \\ & \leq \frac{2}{\tau \Delta E} \sum_{E, E'} |\langle E | \text{col } 1 \rangle|^2 |\langle E' | \text{col } 2n \rangle|^2 = \frac{2}{\tau \Delta E}. \end{aligned} \quad (48)$$

Thus we have

$$\frac{1}{\tau} \int_0^\tau dt |\langle \text{col } 2n | e^{-iHt} | \text{col } 1 \rangle|^2 \geq \frac{1}{2n} - \frac{2}{\tau \Delta E} \geq \frac{1}{2n} (1 - \epsilon) \quad (49)$$

where the last inequality follows since $\tau \geq \frac{4n}{\epsilon \Delta E}$ by assumption. ■

Now we need to prove that the minimum gap ΔE is only polynomially small.

Lemma 2 *The smallest gap between any pair of eigenvalues of the Hamiltonian satisfies*

$$\Delta E > \frac{2\pi^2}{(1 + \sqrt{2})n^3} + O(1/n^4). \quad (50)$$

Proof To evaluate the spacings between eigenvalues, we need to use the quantization condition. We have

$$\langle \text{col } n | H | E \rangle = 2 \cos p \langle \text{col } n | E \rangle \quad (51)$$

so that

$$\sqrt{2} \langle \text{col}(n+1) | E \rangle + \langle \text{col}(n-1) | E \rangle = 2 \cos p \langle \text{col } n | E \rangle \quad (52)$$

and using (44), we have

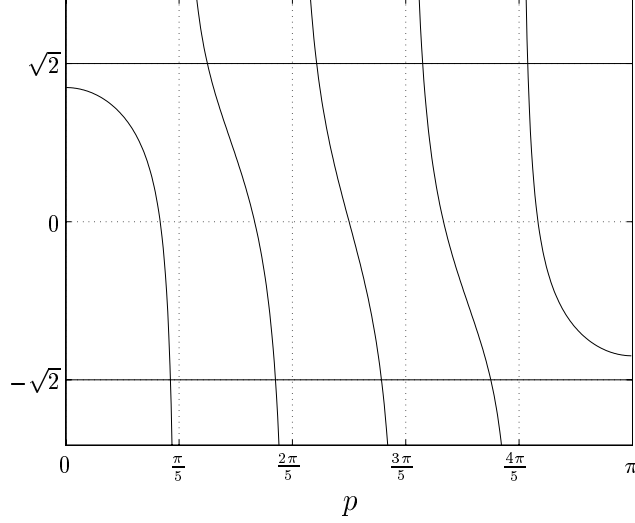
$$\pm \sqrt{2} \sin np + \sin((n-1)p) = 2 \cos p \sin np \quad (53)$$

which simplifies to

$$\frac{\sin((n+1)p)}{\sin np} = \pm \sqrt{2}. \quad (54)$$

In Figure 6 we plot the left hand side of (54) for $n = 5$. The intersections with $-\sqrt{2}$ occur to the left of the zeros of $\sin np$, which occur at $\pi l/n$ for $l = 1, 2, \dots, n-1$. For the values of p that intersect $-\sqrt{2}$, we can write $p = (\pi l/n) - \delta$. Equation (54) with $-\sqrt{2}$ on the right hand side is now

$$-\sqrt{2} \sin n\delta = \sin \left(n\delta - \frac{l\pi}{n} + \delta \right). \quad (55)$$

FIG. 6: Left hand side of (54) for $n = 5$.

Write $\delta = (c/n) + (d/n^2) + O(1/n^3)$. Taking $n \rightarrow \infty$ in (55) gives $-\sqrt{2} \sin c = \sin c$, which implies that $c = 0$. We then get

$$-\sqrt{2} \sin \left(\frac{d}{n} + O(1/n^2) \right) = \sin \left(\frac{d}{n} - \frac{l\pi}{n} + O(1/n^2) \right) \quad (56)$$

which gives, as $n \rightarrow \infty$,

$$d = \frac{l\pi}{1 + \sqrt{2}}. \quad (57)$$

Thus we have that the roots of (54) with $-\sqrt{2}$ on the right hand side are of the form

$$p = \frac{l\pi}{n} - \frac{l\pi}{(1 + \sqrt{2})n^2} + O(1/n^3). \quad (58)$$

Let p' and p'' be the two roots of (54) closest to the root p just found, with $p' < p < p''$. From the figure we see that p' and p'' both are roots of (54) with $+\sqrt{2}$. (Note that the smallest p , corresponding to $l = 1$, does not have a p' .) We see that p'' lies to the right of the zero of $\sin np$ at $p = l\pi/n$. We also see that p' lies to the left of the zero of $\sin((n+1)p)$ at $l\pi/(n+1)$. Therefore we have

$$p' < \frac{l\pi}{n} - \frac{l\pi}{n^2} + O(1/n^3) \quad (59)$$

$$p'' > \frac{l\pi}{n}, \quad (60)$$

from which we conclude that

$$p - p' > \frac{l\pi\sqrt{2}}{(1 + \sqrt{2})n^2} + O(1/n^3), \quad l = 2, 3, \dots, n-1 \quad (61)$$

$$p'' - p > \frac{l\pi}{(1 + \sqrt{2})n^2} + O(1/n^3), \quad l = 1, 2, \dots, n-1. \quad (62)$$

Thus the smallest spacing is at least $\pi/[(1 + \sqrt{2})n^2] + O(1/n^3)$.

Now for a given p , the corresponding eigenvalue is $2 \cos p$. For small Δp , the spacing ΔE is related to the spacing Δp by

$$\Delta E = 2|\Delta p \sin p| + O((\Delta p)^2) . \quad (63)$$

The factor $\sin p = \sin(l\pi/n + O(1/n^2))$ is smallest when $l = 1$, so we have

$$\Delta E > \frac{2\pi^2}{(1 + \sqrt{2})n^3} + O(1/n^4) > \frac{8}{n^3} \text{ for } n \text{ sufficiently large.} \quad (64)$$

The alert reader will note from the figure with $n = 5$ that there are only 8 roots, whereas the dimension of the reduced space is 10, so there are actually 10 eigenvalues. In general, there are $n - 2$ roots of (54) with p real. If we let $p = ik$ with k real, we can have eigenstates of the form (44) with $\sin pj$ replaced by $\sinh kj$ and $\pm \sin(p(2n + 1 - j))$ replaced by $\pm \sinh(k(2n + 1 - j))$. The corresponding eigenvalue is $2 \cosh k$ and the condition (54) becomes

$$\frac{\sinh((n + 1)k)}{\sinh nk} = \pm \sqrt{2}. \quad (65)$$

As $n \rightarrow \infty$, the root of this equation is at $e^k = \sqrt{2}$, which corresponds to an eigenvalue $\sqrt{2} + \frac{1}{\sqrt{2}}$. To obtain the last eigenvalue, let $p = \pi + ik$. The eigenvalue is then $-2 \cosh k$. The quantization condition is now the same as (65), and as $n \rightarrow \infty$, the eigenvalue is $-(\sqrt{2} + \frac{1}{\sqrt{2}})$. So we have found two eigenvalues at $\pm(\sqrt{2} + \frac{1}{\sqrt{2}})$ with corrections that vanish exponentially as $n \rightarrow \infty$. Since the other $n - 2$ eigenvalues are all in the range $[-2, 2]$, our conclusion about the minimum spacing is unchanged. ■

Using Lemma 1 and Lemma 2, we find

Theorem 3 *For n sufficiently large, running the quantum walk for a time chosen uniformly in $[0, \frac{n^4}{2\epsilon}]$ and then measuring in the computational basis yields a probability of finding the EXIT that is greater than $\frac{1}{2n}(1 - \epsilon)$.*

To summarize, we have presented an efficient algorithm for traversing any graph G'_n using a quantum computer. The computer is prepared in the state corresponding to the ENTRANCE, and the quantum walk is simulated using the construction described in Section III B. After running the walk for a certain time t , the state of the computer is measured in the computational basis. The oracle can then be used to check whether the resulting vertex name corresponds to a vertex of degree 2 other than ENTRANCE, in which case it must be EXIT. Theorem 3 shows that by choosing an appropriate $t = \text{poly}(n)$, the probability of finding the name of the EXIT can be $O(1/n)$. By repeating this process $\text{poly}(n)$ times, the success probability can be made arbitrarily close to 1. Combining this with the efficient implementation of the quantum walk described in Section III B, we see that the quantum walk algorithm finds the name of the EXIT with high probability using $\text{poly}(n)$ calls to the oracle.

IV. CLASSICAL LOWER BOUND

We now prove that no classical algorithm can find the EXIT with high probability in subexponential time. We do this by considering a series of games and proving relations between them. The first game is equivalent to our problem, and each new game will be essentially as easy to win. Finally, we will show that the easiest game cannot be won in subexponential time.

Until now, we have not defined the specific coloring of the edges of the graph. We did not need to consider a particular coloring in Section III because the quantum algorithm works given *any* consistent coloring. However, to prove the classical lower bound, we need to specify a coloring that does not supply information about the graph. The coloring is chosen at random as follows. At each vertex in an even numbered column, randomly color the incident edges A, B, C . At each vertex in an odd numbered column, randomly append 1, 2, 3 to the colors of the incident edges. The edges are then colored with one of nine colors, $A1, A2, A3, B1, \dots, C3$. Because of the structure of G'_n , any such coloring is consistent, but as we will see below, it does not provide any useful information to a classical algorithm.

Our problem is equivalent to the following game:

Game 1 *The oracle contains a random set of names for the vertices of the randomly chosen graph G'_n such that each vertex has a distinct $2n$ -bit string as its name and the ENTRANCE vertex has the name 0. In addition, the edges of the graph are randomly colored as described above. At each step, the algorithm sends a $2n$ -bit string to the oracle, and if there exists a vertex with that name, the oracle returns the names of the neighbors of that vertex and the colors of the edges that join them. The algorithm wins if it ever sends the oracle the name of the EXIT vertex.*

Note that there are three sources of randomness in this oracle: in the choice of a graph G'_n , in the random naming of its vertices, and in the random coloring of its edges. We first consider a fixed graph G and coloring C and only draw implications from the random names. Throughout this section, G always refers to one of the graphs G'_n . For a game X with a graph G and a coloring C , the success probability of the algorithm A is defined as

$$\mathbb{P}_X^{G,C}(A) = \Pr_{\text{names}} [\text{algorithm } A \text{ wins game } X \text{ on graph } G \text{ with coloring } C] \quad (66)$$

where $\Pr_{\text{names}} [\cdot]$ means the probability is taken over the random naming of vertices.

In Game 1, the algorithm could traverse a disconnected subgraph of G'_n . But because there are exponentially many more strings of $2n$ bits than vertices in the graph, it is highly unlikely that any algorithm will ever guess the name of a vertex that it was not sent by the oracle. Thus, Game 1 is essentially equivalent to the following game:

Game 2 *The oracle contains a graph, set of vertex names, and edge coloring as described in Game 1. At each step, the algorithm sends the oracle the name of the ENTRANCE vertex or the name of a vertex it has previously been sent by the oracle. The oracle then returns the names of the neighbors of that vertex and the colors of the edges that join them. The algorithm wins if it ever sends the oracle the name of the EXIT vertex.*

The next lemma shows that, if the algorithms run for a sufficiently short time, then the success probabilities for Game 1 and Game 2 can only differ by a small amount.

Lemma 4 *For every algorithm A for Game 1 that makes at most t oracle queries, there exists an algorithm A' for Game 2 that also makes at most t oracle queries such that for all graphs G and all colorings C ,*

$$\mathbb{P}_1^{G,C}(A) \leq \mathbb{P}_2^{G,C}(A') + O(t/2^n). \quad (67)$$

Proof Algorithm A' simulates A , but whenever A queries a name it has not previously been sent by the oracle, A' assumes the result of the query is $11\dots 1$. The chance that A can discover the name of a vertex that it is not told by the oracle is at most $t(2^{n+2}-2)/2^{2n}$, and unless this happens, the two algorithms will have similar behavior. ■

Having restricted the algorithm to traverse a connected subgraph, we will now show how to eliminate the coloring. Consider the following game, which is the same as Game 2 except that the oracle does not provide a coloring:

Game 3 *The oracle contains a graph and a set of vertex names as described in Game 1. At each step, the algorithm sends the oracle the name of the ENTRANCE vertex or the name of a vertex it has previously been sent by the oracle. The oracle then returns the names of the neighbors of that vertex. The algorithm wins if ever sends the oracle the name of the EXIT vertex.*

For games that do not involve a coloring, we define the success probability as

$$\mathbb{P}_X^G(A) = \Pr_{\text{names}} [\text{algorithm } A \text{ wins game } X \text{ on graph } G]. \quad (68)$$

The next lemma shows that Game 2 and Game 3 are equivalent:

Lemma 5 *For any algorithm A for Game 2 that makes t queries to the oracle, there is an algorithm A' for Game 3 that also makes at most t queries to the oracle such that for all graphs G ,*

$$\mathbb{E}_C \left[\mathbb{P}_2^{G,C}(A) \right] = \mathbb{P}_3^G(A'). \quad (69)$$

Here $\mathbb{E}_C [\cdot]$ means the expectation is taken over the random coloring of edges.

Proof The only information needed to make up a coloring is whether a vertex is in an even or odd column, and this information is always available to an algorithm that can only traverse a connected subgraph. Thus A' can make up its own random coloring as it goes, and have the same probability of success as A . ■

To obtain a bound on the success probability of Game 3, we will compare it with a simpler game, which is the same except that it provides an additional way to win:

Game 4 *The oracle contains a graph and a set of vertex names as described in Game 1. At each step, the algorithm and the oracle interact as in Game 3. The algorithm wins if ever sends the oracle the name of the EXIT vertex, or if the subgraph it has seen contains a cycle.*

Game 4 is clearly easier to win than Game 3, so we have

Lemma 6 *For all algorithms A for Game 3,*

$$\mathbb{P}_3^G(A) \leq \mathbb{P}_4^G(A). \quad (70)$$

Now we further restrict the form of the subgraph that can be seen by the algorithm unless it wins Game 4. We will show that the subgraph an algorithm sees must be a random embedding of a rooted binary tree. For a rooted binary tree T , we define an embedding of T into G to be a function π from the vertices of T to the vertices of G such that $\pi(\text{ROOT}) = \text{ENTRANCE}$ and for all vertices u and v that are neighbors in T , $\pi(u)$ and $\pi(v)$ are neighbors in G . We say that an embedding of T is *proper* if $\pi(u) \neq \pi(v)$ for $u \neq v$. We say that a tree T *exits* under an embedding π if $\pi(v) = \text{EXIT}$ for some $v \in T$.

We must specify what we mean by a random embedding of a tree. Intuitively, a random embedding of a tree is obtained by setting $\pi(\text{ROOT}) = \text{ENTRANCE}$ and then mapping the rest of T into G at random. We define this formally for trees T in which each internal vertex has two children (it will not be necessary to consider others). A random embedding is obtained as follows:

1. Label the ROOT of T as 0, and label the other vertices of T with consecutive integers so that if vertex i lies on the path from the root to vertex j , then i is less than j .
2. Set $\pi(0) = \text{ENTRANCE}$.
3. Let i and j be the neighbors of 0 in T .
4. Let u and v be the neighbors of ENTRANCE in G .
5. With probability 1/2 set $\pi(i) = u$ and $\pi(j) = v$, and with probability 1/2 set $\pi(i) = v$ and $\pi(j) = u$.
6. For $i = 1, 2, 3, \dots$, if vertex i is not a leaf, and $\pi(i)$ is not EXIT or ENTRANCE,
 - (a) Let j and k denote the children of vertex i , and let l denote the parent of vertex i .
 - (b) Let u and v be the neighbors of $\pi(i)$ in G other than $\pi(l)$.
 - (c) With probability 1/2 set $\pi(i) = u$ and $\pi(j) = v$, and with probability 1/2 set $\pi(i) = v$ and $\pi(j) = u$.

We can now define the game of finding a tree T for which a randomly chosen π is an improper embedding or T exits:

Game 5 *The algorithm outputs a rooted binary tree T with t vertices in which each internal vertex has two children. A random π is chosen. The algorithm wins if π is an improper embedding of T in G'_n or if T exits G'_n under π .*

As the algorithm A merely serves to produce a distribution on trees T , we define

$$\mathbb{P}^G(T) = \mathbb{P}_\pi [\pi \text{ is an improper embedding of } T \text{ or } T \text{ exits } G \text{ under } \pi], \quad (71)$$

and observe that for every distribution on graphs G and all algorithms taking at most t steps,

$$\max_A \mathbb{E}_G [\mathbb{P}_5^G(A)] \leq \max_{\text{trees } T \text{ with } t \text{ vertices}} \mathbb{E}_G [\mathbb{P}^G(T)]. \quad (72)$$

Game 4 and Game 5 are also equivalent:

Lemma 7 *For any algorithm A for Game 4 that uses at most t queries of the oracle, there exists an algorithm A' for Game 5 that outputs a tree of at most t vertices such that for all graphs G ,*

$$\mathbb{P}_4^G(A) = \mathbb{P}_5^G(A'). \quad (73)$$

Proof Algorithm A halts if it ever finds a cycle, exits, or uses t steps. Algorithm A' will generate a (random) tree by simulating A . Suppose that vertex a in graph G corresponds to vertex a' in the tree that A' is generating. If A asks the oracle for the names of the neighbors of a , A' generates two unused names b' and c' at random and uses them as the neighbors of a' . Now b' and c' correspond to b and c , the neighbors of a in G . Using the tree generated by A' in Game 5 has the same behavior as using A in Game 4. ■

Finally, we bound the probability that an algorithm wins Game 5:

Lemma 8 For rooted trees T of at most $2^{n/6}$ vertices,

$$\max_T \mathbb{E}_G [\mathbb{P}^G(T)] \leq 3 \cdot 2^{-n/6}. \quad (74)$$

Proof Let T be a tree with t vertices, $t \leq 2^{n/6}$, with image $\pi(T)$ in G'_n under the random embedding π . The vertices of columns $n+1, n+2, \dots, n + \frac{n}{2}$ in G'_n divide naturally into $2^{n/2}$ complete binary subtrees of height $n/2$.

- (i) It is very unlikely that $\pi(T)$ contains the root of any of these subtrees, i.e., that $\pi(T)$ includes any vertex in column $n + \frac{n}{2}$. Consider a path in T from the root to a leaf. The path has length at most t , and there are at most t such paths. To reach column $n + \frac{n}{2}$ from column $n+1$, π must choose to move right $\frac{n}{2} - 1$ times in a row, which has probability $2^{1-n/2}$. Since there are at most t tries on each path of T (from the root to a leaf) and there are at most t such paths, the probability is bounded by $t^2 \cdot 2^{1-n/2}$.
- (ii) If $\pi(T)$ contains a cycle, then there are two vertices a, b in T such that $\pi(a) = \pi(b)$. Let P be the path in T from a to b . Then $\pi(P)$ is a cycle in G'_n . Let c be the vertex in T closest to the root on the path $\pi(P)$, and let $\pi(P)$ consist of the path $\pi(P_1)$ from c to a and $\pi(P_2)$ from c to b .

Let $S_1, S_2, \dots, S_{2^{n/2}}$ denote the $2^{n/2}$ subtrees described above. Let $S'_1, S'_2, \dots, S'_{2^{n/2}}$ denote the corresponding subtrees made out of columns $\frac{n}{2} + 1$ to n . Without loss of generality, let $\pi(c)$ be in the left tree of G'_n , i.e., in a column $\leq n$, as shown in Figure 7.

Now $\pi(P_1)$ visits a sequence of subtrees $S'_{i_1}, S'_{j_1}, S'_{i_2}, \dots$. Similarly $\pi(P_2)$ visits a sequence of subtrees $S'_{k_1}, S'_{l_1}, S'_{k_2}, \dots$. Since $\pi(a) = \pi(b)$, the last subtree on these two lists must be the same. (The other possibility is that $\pi(a) = \pi(b)$ does not lie in any subtree, hence lies in columns 1 through $\frac{n}{2}$ or $n + \frac{n}{2} + 1$ through $2n$. But the event that column $n + \frac{n}{2}$ is ever reached has already been shown to be unlikely in part (i). The same argument bounds the probability of a return to column $\frac{n}{2}$ after a visit to column $n+1$.) At least one of the lists has more than one term (or all vertices visited are in the left tree, which can't make a cycle). The probability that the last terms on the two lists agree is bounded by $2^{n/2}/(2^n - t)$, by the construction of the random cycle that connected the two trees of G'_n . As long as $t \leq 2^{n-1}$, we have $2^{n/2}/(2^n - t) \leq 2 \cdot 2^{-n/2}$. Since there are $\binom{t}{2}$ paths P to be considered, the probability of a cycle is less than $t^2 \cdot 2^{-n/2}$.

Overall we have shown that

$$\mathbb{E}_G [\mathbb{P}^G(T)] \leq t^2 \cdot 2^{-n/2} + t^2 \cdot 2^{1-n/2} \quad (75)$$

$$\leq 3 \cdot 2^{-n/6} \quad (76)$$

if $t \leq 2^{n/6}$. ■

We have proved the following.

Theorem 9 Any classical algorithm that makes at most $2^{n/6}$ queries to the oracle finds the EXIT with probability at most $4 \cdot 2^{-n/6}$.



FIG. 7: Graphical representation of part (ii) of the proof of Lemma 8. The triangles represent the subtrees of G'_n of height $n/2$, the dashed line represents the path $\pi(P_1)$, and the dotted line represents the path $\pi(P_2)$. Together, these paths form a cycle in the graph.

V. DISCUSSION

In this paper, we have applied a general implementation of (continuous time) quantum walks to an oracular problem, and we have proved that this problem can be solved efficiently by a quantum computer, whereas no classical algorithm can do the same. We considered the problem of finding the name of the EXIT of a particular graph starting from the ENTRANCE, given an oracle that outputs the names of the neighbors of an input vertex. Note that although our algorithm finds the name of the EXIT, it does not find a particular path from ENTRANCE to EXIT.

We can also view our problem as a decision problem by asking whether the first bit of the name of the EXIT is 0 or 1. Therefore, our result can be also interpreted as showing the existence of a new kind of oracle relative to which $\text{BQP} \neq \text{BPP}$ [3, 4]. We also note that, although it is convenient to express our results in terms of a graph traversal problem, the results can also be cast in terms of a graph reachability problem, where one is given a graph and two vertices and the goal is to determine whether there is a path connecting those vertices.

To efficiently simulate the quantum walk on a general graph, we required a coloring of its edges. In the specific case of a quantum walk on a bipartite graph starting at the ENTRANCE, the walk can be constructed without a coloring, as we describe in Appendix B. It is an open question whether one can find other techniques for implementing quantum walks. In particular, we do not know whether it is possible to efficiently simulate a quantum walk on an arbitrary uncolored graph

without any restriction on the structure of the graph or the initial state.

Many computational problems can be recast as determining some property of a graph. A natural question is whether there are other interesting computational problems (especially non-oracular ones) that are classically hard (or are believed to be classically hard) but that can be solved efficiently on a quantum computer employing quantum walks.

Acknowledgments

We thank Jeffrey Goldstone for helpful discussions and encouragement throughout this project. We also thank Dorit Aharonov for stimulating correspondence about the implementation of quantum walks and John Watrous for helpful remarks about quantum walks.

AMC received support from the Fannie and John Hertz Foundation, RC was supported in part by Canada's NSERC, ED received support from the Istituto Nazionale di Fisica Nucleare (Italy), and DAS was supported in part by an Alfred P. Sloan Foundation Fellowship and NSF Award CCR-0112487. This work was supported in part by the Cambridge–MIT Foundation, by the Department of Energy under cooperative research agreement DE-FC02-94ER40818, and by the National Security Agency and Advanced Research and Development Activity under Army Research Office contract DAAD19-01-1-0656.

APPENDIX A: AN EFFICIENT CLASSICAL ALGORITHM TO TRAVERSE THE HYPERCUBE

In this appendix we describe a classical algorithm that traverses the hypercube graph, where ENTRANCE is any vertex and EXIT is defined to be the unique vertex whose distance from ENTRANCE is n .

The idea of the algorithm is to start at ENTRANCE and repeatedly move one level closer to EXIT, where level k is the set of vertices whose minimum distance from ENTRANCE is k . In other words, the algorithm will construct a sequence of vertices $a_0 = \text{ENTRANCE}, a_1, \dots, a_{n-1}, a_n = \text{EXIT}$, where a_k is in level k . To choose the vertex a_{k+1} , the algorithm will also require a set of vertices S_{k-1} , which is the set of all neighbors of a_k that are at level $k - 1$. Note that every neighbor of a_k is either at level $k - 1$ or $k + 1$.

The algorithm proceeds as follows. First, let a_1 be any neighbor of a_0 and let $S_0 = \{a_0\}$. Now suppose that (for some k) a_k and S_{k-1} are known. Then let a_{k+1} be any neighbor of a_k that is not in S_{k-1} , and let S_k contain every neighbor of a_{k+1} that is also a neighbor of some element of S_{k-1} . The key point is that the set S_k constructed in this way is in fact the set of all neighbors of a_{k+1} whose distance from a_0 is k . After n such steps, the algorithm reaches $a_n = \text{EXIT}$. Each step takes $O(n)$ queries, so the total number of queries is $O(n^2)$.

APPENDIX B: IMPLEMENTATION OF THE QUANTUM WALK STARTING FROM A PARTICULAR VERTEX OF A BIPARTITE GRAPH

In this appendix we outline an implementation of the quantum walk on a bipartite graph with the initial state given by ENTRANCE in the case where the oracle does not provide an edge coloring. That is, the oracle is still given by (6), but now $v_c(v_c(a))$ need not equal a .

This implementation uses the fact that, in any case, the neighbors of a vertex must be given in *some* order, and this provides a coloring of the directed edges. For a bipartite graph, this can be

used to construct a consistent coloring of the undirected graph. For our particular graphs G'_n , the construction is similar to the nine color construction found in Section IV.

The construction of the coloring requires a parity bit indicating which side of the bipartite graph a vertex is in, and thus essentially requires that the initial state is a particular vertex (e.g., the ENTRANCE) and not a general superposition. The ENTRANCE is defined to be on side 0 of the bipartition, so the initial state has the parity bit equal to 0. By flipping the value of the parity bit each time the oracle is queried, the algorithm can keep track of the parity of each vertex. In other words, the name of each vertex can be assumed to contain a special bit that indicates which side of the bipartition the vertex is on.

Using the parity bit, we can combine the colors of the directed edges to form a consistent coloring of the undirected edges. For vertices with parity 0, the color of each adjacent undirected edge is (incoming color, outgoing color), and for vertices with parity 1, the color of each adjacent undirected edge is (outgoing color, incoming color). The color of an adjacent undirected edge can be easily computed using the oracle and the parity bit, and it is also easy to compute whether there is an undirected edge of a particular color incident on a given vertex. Thus, the oracle that provides this consistent coloring of the undirected edges can be simulated using the oracle that does not provide an edge coloring.

-
- [1] D. Deutsch, *Quantum theory, the Church-Turing principle, and the universal quantum computer*, Proc. Roy. Soc. London A **400**, 97 (1985).
 - [2] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proc. Roy. Soc. London A **439**, 553 (1992).
 - [3] E. Bernstein and U. Vazirani, *Quantum complexity theory*, Proc. 25th ACM Symposium on the Theory of Computing, 11 (1993).
 - [4] D. Simon, *On the power of quantum computation*, Proc. 35th IEEE Symposium on the Foundations of Computer Science, 116 (1994).
 - [5] P. W. Shor, *Algorithms for quantum computation: discrete logarithms and factoring*, Proc. 35th IEEE Symposium on Foundations of Computer Science, 124 (1994).
 - [6] A. Kitaev, *Quantum measurements and the abelian stabilizer problem*, quant-ph/9511026.
 - [7] M. Mosca and A. Ekert, *The hidden subgroup problem and eigenvalue estimation on a quantum computer*, Proc. 1st NASA International Conference on Quantum Computing and Quantum Communication, Vol. 1509 of *Lecture Notes in Computer Science* (1999).
 - [8] J. N. de Beaudrap, R. Cleve, and J. Watrous, *Sharp quantum vs. classical query complexity separations*, quant-ph/0011065.
 - [9] W. van Dam and S. Hallgren, *Efficient quantum algorithms for shifted quadratic character problems*, quant-ph/0011067.
 - [10] S. Hallgren, A. Russell, and A. Ta-Shma, *Normal subgroup reconstruction and quantum computation using group representations*, Proc. 32nd ACM Symposium on the Theory of Computing, 627 (2000).
 - [11] W. van Dam, S. Hallgren, and L. Ip, *Quantum algorithms for hidden coset problems*, unpublished.
 - [12] M. Grigni, L. Schulman, M. Vazirani, and U. Vazirani, *Quantum mechanical algorithms for the non-abelian hidden subgroup problem*, Proc. 33rd ACM Symposium on the Theory of Computing, 68 (2001).
 - [13] G. Ivanos, F. Magniez, and M. Santha, *Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem*, quant-ph/0102014.
 - [14] J. Watrous, *Quantum algorithms for solvable groups*, Proc. 33rd ACM Symposium on the Theory of Computing, 60 (2001).
 - [15] S. Hallgren, *Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem*, Proc. 34th ACM Symposium on the Theory of Computing, 653 (2002).
 - [16] E. Farhi and S. Gutmann, *Quantum computation and decision trees*, Phys. Rev. A **58**, 915 (1998).
 - [17] A. M. Childs, E. Farhi, and S. Gutmann, *An example of the difference between quantum and classical random walks*, Quantum Information Processing **1**, 35 (2002).

- [18] Y. Aharonov, L. Davidovich, and N. Zagury, *Quantum random walks*, Phys. Rev. A **48**, 1687 (1993).
- [19] D. A. Meyer, *From quantum cellular automata to quantum lattice gasses*, J. Stat. Phys. **85**, 551 (1996).
- [20] J. Watrous, *Quantum simulations of classical random walks and undirected graph connectivity*, J. Computer and System Sciences **62**, 376 (2001).
- [21] D. Aharonov, A. Ambainis, J. Kempe, and U. Vazirani, *Quantum walks on graphs*, in Proceedings of the 33rd ACM Symposium on the Theory of Computing, 50 (ACM Press, New York, 2001).
- [22] A. Ambainis, E. Bach, A. Nayak, A. Vishwanath, and J. Watrous, *One-dimensional quantum walks*, Proc. 33rd ACM Symposium on the Theory of Computing, 37 (ACM Press, New York, 2001).
- [23] C. Moore and A. Russell, *Quantum walks on the hypercube*, quant-ph/0104137.
- [24] J. Kempe, *Quantum random walks hit exponentially faster*, quant-ph/0205083.
- [25] V. G. Vizing, *On an estimate of the chromatic class of a p -graph*, Diskret. Analiz **3**, 23 (1964).
- [26] S. Lloyd, *Universal quantum simulators*, Science **273**, 1073 (1996).
- [27] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2000).
- [28] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions* (Dover, New York, 1972).
- [29] J. Goldstone, personal communication, September 2002.