Exponential Algorithmic Speedup by a Quantum Walk

Andrew M. Childs* Richard Cleve† Enrico Deotto* Edward Farhi*
Sam Gutmann‡ Daniel A. Spielman§

ABSTRACT

We construct a black box graph traversal 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 black box setting. We then show how this quantum walk solves our problem by rapidly traversing a graph. Finally, we prove that no classical algorithm can solve the problem in subexponential time.

Categories and Subject Descriptors

F. [**Theory of Computation**]: Computation by Abstract Devices

General Terms

Theory

Keywords

Quantum algorithms, quantum walks

1. 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 a black box problem exponentially faster than any classical algorithm.

Black box problems provided the first examples of algorithmic speedup using quantum instead of classical computers. Deutsch gave an example of a problem [10] that can be solved on a quantum computer using one query, but that requires two queries on a classical computer. This was followed by a series of black box problems [11, 6, 26] with increasingly strong quantum versus classical query complexity separations—including exponential ones. These results contributed to Shor's discovery of polynomial-time algorithms for integer factorization and discrete logarithms [25]. A number of generalizations and variations of these algorithms have been discovered for solving both black box and conventional problems with exponential speedup (e.g., [19, 23, 5, 8, 16, 9, 14, 17, 28, 15]). All of them are fundamentally based on quantum Fourier transforms.

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 [12], who proposed the model of a quantum walk used in the present paper. They gave an example of a graph in which the time to propagate between two vertices is exponentially faster for the quantum walk than for the corresponding classical random walk. A simpler example was given in [7]. However, as we explain in Section 2, these results do not imply algorithmic speedup. In the present paper, we modify the example presented in [7] and construct a black box problem based on it that can be solved efficiently a quantum algorithm that simulates a quantum walk, whereas no classical algorithm can solve the problem in subexponential time. Although the

*Center for Theoretical Physics, Massachusetts Institute of Technology, Cambridge, MA 02139. amchilds@mit.edu, deotto@mitlns.mit.edu, farhi@mit.edu
†Dept. of Computer Science, Univ. of Calgary, Calgary, Alberta, Canada T2N 1N4. cleve@cpsc.ucalgary.ca
‡Dept. of Mathematics, Northeastern University, Boston, MA 02115. sgutm@neu.edu
§Dept. of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139. spielman@math.mit.edu

STOC'03, June 9–11, 2003, San Diego, California, USA.

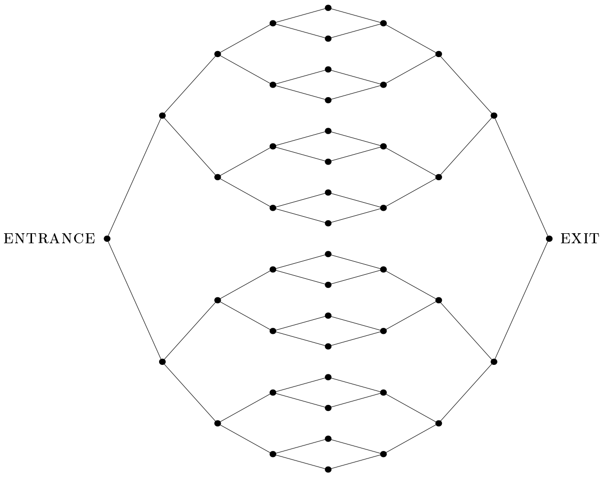

Figure 1: The graph G_4.

quantum walk is defined as a continuous time process, our quantum algorithm simulates a good approximation of it in the conventional quantum circuit model.

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") [21, 29, 2, 4]. 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 2, we define our problem. In Section 3, we present a quantum algorithm for solving this problem in polynomial time with high probability, and in Section 4, we show that the problem cannot be solved classically in subexponential time with non-negligible probability. We conclude with a discussion of the results in Section 5.

2. THE GRAPH TRAVERSAL PROBLEM

In this section we describe in detail the problem we address in this paper. The problem involves determining a property of a certain type of graph whose structure is given in the form of black box.

Our problem is based on a generalization of the graphs G_n presented in [7], which consist 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, a 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 a black box framework in which graphs can be specified, and where our notion of an algorithm traversing a graph can be made precise.

Let $G = (V(G), E(G))$ be a graph with N vertices. To represent G as a black box, let m be such that $2^m > N$ and let k be at least as large as the maximum vertex degree in G. Assign each vertex $a \in V(G)$ a distinct m-bit string as its name, except do not assign $11\ldots1$ as the name of any vertex. For each vertex $a \in V(G)$, assign the outgoing edges of a labels from a set L of size k. For $a \in \{0,1\}^m$ and $c \in L$, define $v_c(a)$ as the adjacent vertex reached by following the outgoing edge of a labeled by c, if such an edge exists. If no such edge exists or if $a \notin V(G)$ then $v_c(a) = 11\ldots1$. The resulting black box for G takes $c \in L$ and $a \in \{0,1\}^m$ as input and returns the value of $v_c(a)$. For quantum algorithms, this operation is defined as a unitary transformation U in the usual way. That is, for $a, b \in \{0,1\}^m$ and $c \in L$,

$$U|c, a, b\rangle = |c, a, b \oplus v_c(a)\rangle, \qquad (1)$$

where $\oplus$ denotes bitwise addition modulo 2.

We now define a notion of traversing a graph G from its ENTRANCE to its EXIT.

DEFINITION 1. *Let G be a graph and* ENTRANCE *and* EXIT *be two vertices of G. The input of the* traversal problem *is a black box for G and the name of the* ENTRANCE. *The output is the name of the* EXIT.

For the graphs G_n, set ENTRANCE and EXIT to the left root and the right root of the trees, respectively. This instance of the traversal problem can be solved in time polynomial in n 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 s steps we are again at a central vertex, we know that the wrong move happened after $s/2$ steps (s 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.[1]

We now describe how the graphs G_n are modified so that they cannot be traversed efficiently by any classical algorithm. We choose a graph G'_n at random from a particular

[1] Also, there is a polynomial-time classical traversal algorithm for the n-dimensional hypercube (where ENTRANCE and EXIT are two vertices whose distance apart 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-black box setting [22, 18].

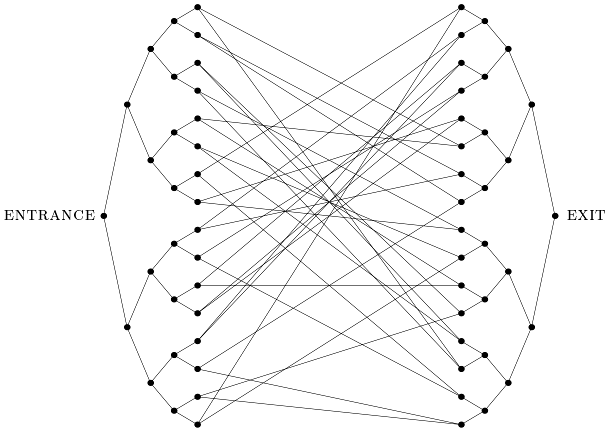

Figure 2: A typical graph G'_4.

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, and so on, until every leaf on the left is connected to two leaves on the right (and vice versa).

In Section 3, we describe a quantum algorithm that solves the graph traversal problem for G'_n in polynomial-time, and, in Section 4, we show that any classical algorithm that solves this traversal problem with nonnegligible probability for a randomly generated black box for G'_n takes exponential time.

3. QUANTUM ALGORITHM

In this section, we give a polynomial-time quantum algorithm based on quantum walks for solving our problem. We begin by describing the model of a continuous time quantum walk on a graph in Section 3.1. Then, in Section 3.2, we explain how to use a quantum computer to efficiently approximate the quantum walk on a graph given by such a black box. Finally, in Section 3.3, we prove that, after executing the quantum walk for a polynomial amount of time, if the state is measured, the probability of finding the name of the EXIT is $\Omega(1/n)$. The occurrence of this event can be checked, since EXIT is the only vertex of degree 2 other than ENTRANCE. Therefore, with $O(n)$ executions of the algorithm, the success probability can be made arbitrarily close to 1.

3.1 Quantum walk

We now define our model of a quantum walk on any undirected graph $G = (V(G), E(G))$ that has no self-loops. Let $d(a)$ denote the degree of vertex a.

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, the probability of jumping to any adjacent vertex in a time ϵ is $\gamma\epsilon$ (in the limit as $\epsilon \to 0$). If the graph has N vertices, this classical random walk can be described by the $N \times N$ infinitesimal generator matrix K defined by

$$K_{ab} = \begin{cases} \gamma & \text{if } (a,b) \in E(G) \\ -d(a)\gamma & \text{if } a = b \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

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

$$\frac{\mathrm{d}}{\mathrm{d}t}\, p_a(t) = \sum_b K_{ab}\, p_b(t)\,. \tag{3}$$

Since the columns of K sum to 0, an initially normalized distribution remains normalized: $\sum_a p_a(t) = 1$ for all t.

Now consider quantum evolution in the N-dimensional Hilbert space spanned by $V(G)$. Let the state at time t be given by $\sum_a \alpha_a(t)|a\rangle$. If the Hamiltonian is H then the dynamics are determined by the Schrödinger equation,

$$i\frac{\mathrm{d}}{\mathrm{d}t}\, \alpha_a(t) = \sum_b H_{ab}\, \alpha_b(t)\,. \tag{4}$$

Note the similarity between (3) and (4). A natural quantum analogue to the continuous time classical random walk described above is given by setting H to K [12]. In the quantum case, there is no need for the columns of H to sum to zero; we only require H to be Hermitian. For simplicity, we set the diagonal to zero, and define a quantum walk in terms of a Hamiltonian with matrix elements

$$H_{ab} = \begin{cases} \gamma & \text{if } (a,b) \in E(G) \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

In other words, H is the adjacency matrix of the graph times γ. Since H is Hermitian, an initially normalized state remains normalized: $\sum_a |\alpha_a(t)|^2 = 1$ for all t.

3.2 Implementing the quantum walk

In this section, we describe how to implement a quantum walk on a graph G. Our goal is to simulate the unitary evolution e^{-iHt} with H given by (5). We are not given H explicitly. Rather, we have access to a black box for G as defined by (1). A *simulation* of H is a quantum circuit that approximates the unitary transformation e^{-iHt} within some precision ϵ. We are interested in simulations of quantum walks that are *efficient* in the sense that they consist of a number of queries and additional one- and two-qubit gates that are polynomial with respect to m, k, t, and $1/\epsilon$. We have two simulation results, that are stated in Theorem 1 and Theorem 2 (which appear below).

Some standard tools for simulating Hamiltonians are:

1. *Tensor product.* If $H_1, \ldots, H_j$ each act on $O(1)$ qubits and the product of their eigenvalues is efficiently computable, then we can simulate $H_1 \otimes \cdots \otimes H_j$. This can be done with a generalization of a technique discussed in Section 4.7.3 of [24]. We will not

present the general construction, but we show its application to a specific Hamiltonian below. Note that $H_1 \otimes \cdots \otimes H_j$ may be highly nonlocal, so it may look very different from the Hamiltonian of a physical system.

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

3. *Linear combination.* If we can simulate $H_1, \ldots, H_q$ then we can simulate $H_1 + \cdots + H_q$ as a result of the Lie product formula
$$\left\| e^{-i(H_1+\cdots+H_q)t} - \left(e^{-iH_1 t/r} \cdots e^{-iH_q t/r}\right)^r \right\| = O\left(\frac{qlt^2}{r}\right),$$
where $l = \max_{p,q} \|[H_p, H_q]\|$, the largest norm of a commutator of two of the Hamiltonians. This is a powerful tool for the simulation of *physical* quantum systems, whose Hamiltonians can typically be expressed as sums of local terms [20].

4. *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
$$e^{[H_1,H_2]t} = \lim_{r\to\infty}\left(e^{-iH_1 t/\sqrt{r}} e^{-iH_2 t/\sqrt{r}} e^{iH_1 t/\sqrt{r}} e^{iH_2 t/\sqrt{r}}\right)^r.$$
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.

We will first show how to efficiently simulate the quantum walk on any graph given as a black box provided that the edge labeling is *symmetric*. A symmetric edge labeling is one where $v_c(a) = b$ implies $v_c(b) = a$ for any vertices a and b and any label c. In this case, the labels associated with each edge can be viewed as a coloring of the edges of G. If the maximum degree of G is Δ, then its edges can always be colored with at most $\Delta + 1$ colors [27].

Theorem 1. *Given a symmetrically labeled black box for G, a quantum walk on G can be simulated for time t with precision ϵ by using $O(k^2 t^2/\epsilon)$ black box queries and $O(k^2 t^2 m/\epsilon)$ auxiliary operations.*

Proof. Our quantum walk simulation occurs in the Hilbert space spanned by states of the form $|a, b, r\rangle$, where a and b are m-bit strings and r is a bit. States corresponding to vertices are those of the form $|a, 0, 0\rangle$, for $a \in V(G)$.

We begin by showing how to simulate the evolution generated by the Hermitian operator T satisfying $T|a, b, 0\rangle = |b, a, 0\rangle$ and $T|a, b, 1\rangle = 0$. We will use T as a building block in our simulation of H. Our simulation of T is exact and uses only $O(m)$ one- and two-qubit gates. The operator T may be written as
$$T = S^{(1,m+1)} \otimes S^{(2,m+2)} \otimes \cdots \otimes S^{(m,2m)} \otimes |0\rangle\langle 0| \,, \quad (6)$$

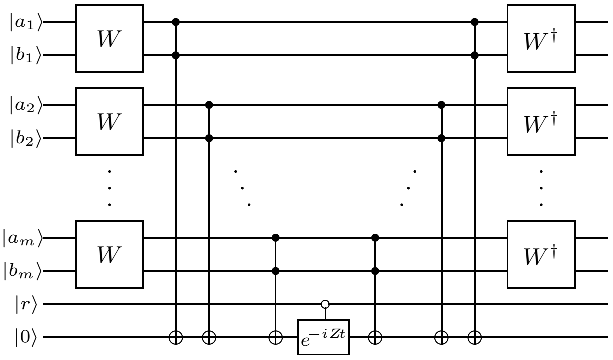

Figure 3: A circuit computing e^{-iTt}.

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 $S|z_1 z_2\rangle = |z_2 z_1\rangle$. The eigenvalues of S are ± 1, and the unique eigenvector with eigenvalue -1 is $\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$, making the eigenvalues of T easy to compute. The circuit shown in Figure 3 computes e^{-iTt} and thus simulates T. In this figure, W denotes the unitary operator such that
$$W^\dagger |z_1 z_2\rangle = \tfrac{1}{\sqrt{2}}(|0 z_2\rangle + (-1)^{z_1} |1 \overline{z}_2\rangle) \,. \quad (7)$$

Applying $W^{\otimes m}$ diagonalizes T, and the Toffoli gates compute the argument of the eigenvalue in an ancilla register initially prepared in the state $|0\rangle$. 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 $Z|z\rangle = (-1)^z |z\rangle$. This controlled phase shift can be performed since it is a two-qubit gate. Finally, we uncompute the eigenvalue and return to the original basis.

We now give our method for implementing the quantum walk on a general graph G for which the labeling is symmetric. First, note that by checking whether $v_c(a) = 11\ldots1$, it is straightforward use a single query to (1) to perform (for each $c \in L$), the transformation
$$V_c |a, b, r\rangle = |a, b \oplus v_c(a), r \oplus f_c(a)\rangle \,, \quad (8)$$
where $f_c(a) = \begin{cases} 0 & v_c(a) \neq 11\ldots1 \\ 1 & v_c(a) = 11\ldots1 \,. \end{cases}$
Also, $V_c^\dagger = V_c$, so by unitary conjugation, we can simulate $V_c^\dagger T V_c$ using two queries and $O(m)$ other gates. Note that, for $a \in V(G)$,
$$\begin{aligned} V_c^\dagger T V_c |a, 0, 0\rangle &= V_c^\dagger T |a, v_c(a), f_c(a)\rangle \\ = \delta_{0,f_c(a)} V_c^\dagger |v_c(a), a, 0\rangle &= \delta_{0,f_c(a)} |v_c(a), 0, 0\rangle, \end{aligned} \quad (9)$$
where δ is the Kronecker delta and we have used the fact that $f_c(a) = 0$ implies $v_c(v_c(a)) = a$ and $f_c(v_c(a)) = 0$.

Finally, by linear combination, with $r \in O(kt^2/\epsilon)$, we can simulate
$$\sum_c V_c^\dagger T V_c \quad (10)$$

with precision ϵ. This is equivalent to H, since, for all $a \in V(G)$, it maps $|a,0,0\rangle$ to $\displaystyle\sum_{c:v_c(a)\in V(G)} |v_c(a),0,0\rangle$. □

The next theorem addresses the case of non-symmetric labelings (where $v_c(v_c(a))$ need not equal a) for a restricted class of graphs and for quantum walks that start at a particular vertex. This subsumes the case of simulating a quantum walk on G'_n starting at vertex ENTRANCE.

THEOREM 2. *If G is bipartite and $a \in V(G)$ then, given any black box for G, a quantum walk on G starting in state $|a\rangle$ can be simulated for time t with precision ϵ by using $O(k^4t^2/\epsilon)$ queries and $O(k^4t^2m/\epsilon)$ auxiliary operations.*

PROOF. The idea is to simulate a symmetric labeling scheme and apply Theorem 1. For each edge (a,b), the labels from its two ends—call them c_a and c_b—are taken together and viewed as a label from a larger set of up to k^2 labels. The bipartite structure of G is used to determine whether this aggregate label is (c_a,c_b) or (c_b,c_a).

The construction makes use of a *parity bit* at each vertex, indicating which side of the bipartite graph it is in. This can be constructed by defining the initial state $|a\rangle$ to have parity bit equal to 0 and then flipping the value of the parity bit each time the black box is queried. Therefore, each vertex can be assumed to contain a parity bit.

If vertex a has parity bit 0 then the label of edge (a,b) is (c_a,c_b); otherwise the label is (c_b,c_a). The label of an incident edge can be easily computed using the black box and the parity bit, and it is also easy to compute whether there is an edge with a particular label incident on a given vertex. What results is a symmetric labeling scheme. □

See [3] for more general Hamiltonian simulations.

3.3 Upper bound on the traversal time

We now consider the quantum walk on the specific family of graphs G'_n of Figure 2, and prove that the walk reaches the EXIT in polynomial time. It can be shown [13] that, at a time of order n, there is a probability of order $n^{-1/3}$ of finding the EXIT if a measurement is made. We shall give a simpler proof of a bound that is not tight—but is nevertheless polynomial.

The analysis of the walk on G'_n reduces 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

$$|C_j\rangle = \frac{1}{\sqrt{N_j}} \sum_{a\in\text{column } j} |a\rangle\,, \tag{11}$$

where N_j is the number of vertices in column j. 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.

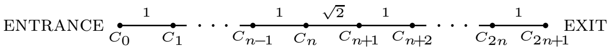

Figure 4: Quantum walk in column subspace of G'_n.

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

$$\langle C_j|H|C_{j+1}\rangle = \begin{cases} \sqrt{2}\gamma & 0 \le j \le 2n \text{ and } j \ne n \\ 2\gamma & j = n \end{cases} \tag{12}$$

(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. 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, the Hamiltonian (12) has non-zero matrix elements

$$\langle C_j|H|C_{j+1}\rangle = \begin{cases} 1 & 1 \le j \le 2n-1 \text{ and } j \ne n \\ \sqrt{2} & j = n\,, \end{cases} \tag{13}$$

(and those deduced by Hermiticity of H).

Define a reflection operator $R|C_j\rangle = |C_{2n+1-j}\rangle$. 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

$$\langle C_j|E\rangle = \begin{cases} \sin pj & 1 \le j \le n \\ \pm\sin(p(2n+1-j)) & n+1 \le j \le 2n, \end{cases} \tag{14}$$

which explicitly vanish at $j=0$ and $j=2n+1$. The eigenvalue corresponding to 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 $|C_1\rangle$ and the EXIT vertex to $|C_{2n}\rangle$.

LEMMA 3. *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 $\tau \ge \frac{4n}{\epsilon\,\Delta E}$ 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

$$\begin{aligned}
&\frac{1}{\tau}\int_0^\tau \mathrm{d}t\,|\langle C_{2n}|e^{-iHt}|C_1\rangle|^2 \\
&\quad= \frac{1}{\tau}\sum_{E,E'}\int_0^\tau \mathrm{d}t\,e^{-i(E-E')t}\langle C_{2n}|E\rangle\langle E|C_1\rangle\langle C_1|E'\rangle\langle E'|C_{2n}\rangle \\
&\quad= \sum_E |\langle E|C_1\rangle|^2|\langle E|C_{2n}\rangle|^2 \\
&\qquad+\sum_{E\ne E'}\frac{1-e^{-i(E-E')\tau}}{i(E-E')\tau}\langle C_{2n}|E\rangle\langle E|C_1\rangle\langle C_1|E'\rangle\langle E'|C_{2n}\rangle.
\end{aligned}$$

By (14), $\langle E|C_1\rangle = \pm\langle E|C_{2n}\rangle$, so the first term is

$$\sum_E |\langle E|C_1\rangle|^4 \geq \frac{1}{2n} \tag{15}$$

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

$$\left|\sum_{E\neq E'} \frac{1-e^{-i(E-E')\tau}}{i(E-E')\tau}\langle C_{2n}|E\rangle\langle E|C_1\rangle\langle C_1|E'\rangle\langle E'|C_{2n}\rangle\right|$$
$$\leq \frac{2}{\tau\Delta E}\sum_{E,E'}|\langle E|C_1\rangle|^2|\langle E'|C_{2n}\rangle|^2 = \frac{2}{\tau\Delta E}\,. \tag{16}$$

Thus we have

$$\frac{1}{\tau}\int_0^\tau \mathrm{d}t\,|\langle C_{2n}|e^{-iHt}|C_1\rangle|^2 \geq \frac{1}{2n} - \frac{2}{\tau\Delta E} \geq \frac{1}{2n}(1-\epsilon) \tag{17}$$

where the last inequality follows since $\tau \geq \frac{4n}{\epsilon\,\Delta E}$. □

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

Lemma 4. *The smallest gap between any pair of eigenvalues of the Hamiltonian satisfies*

$$\Delta E > \frac{2\pi^2}{(1+\sqrt{2})n^3} + O\!\left(\frac{1}{n^4}\right). \tag{18}$$

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

$$\langle C_n|H|E\rangle = 2\cos p\,\langle C_n|E\rangle \tag{19}$$

so that

$$\sqrt{2}\langle C_{n+1}|E\rangle + \langle C_{n-1}|E\rangle = 2\cos p\,\langle C_n|E\rangle \tag{20}$$

and using (14), we have

$$\pm\sqrt{2}\sin np + \sin((n-1)p) = 2\cos p\,\sin np \tag{21}$$

which simplifies to

$$\frac{\sin((n+1)p)}{\sin np} = \pm\sqrt{2}\,. \tag{22}$$

In Figure 5 we plot the left hand side of (22) 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 $l=1,2,\ldots,n-1$. For the values of p that intersect $-\sqrt{2}$, we can write $p=(\pi l/n)-\delta$. Equation (22) with $-\sqrt{2}$ on the right hand side is now

$$-\sqrt{2}\sin n\delta = \sin\!\left(n\delta - \frac{l\pi}{n} + \delta\right)\,. \tag{23}$$

Write $\delta = (c/n) + (d/n^2) + O(1/n^3)$. Taking $n\to\infty$ in (23) gives $-\sqrt{2}\sin c = \sin c$, which implies $c=0$, yielding

$$-\sqrt{2}\sin\!\left(\frac{d}{n}+O\!\left(\frac{1}{n^2}\right)\right) = \sin\!\left(\frac{d}{n}-\frac{l\pi}{n}+O\!\left(\frac{1}{n^2}\right)\right) \tag{24}$$

which gives, as $n\to\infty$, $d=\dfrac{l\pi}{1+\sqrt{2}}$. Thus we have that the roots of (22) with $-\sqrt{2}$ on the right hand side are of the form $p=\dfrac{l\pi}{n}-\dfrac{l\pi}{(1+\sqrt{2})n^2}+O\!\left(\dfrac{1}{n^3}\right)$.

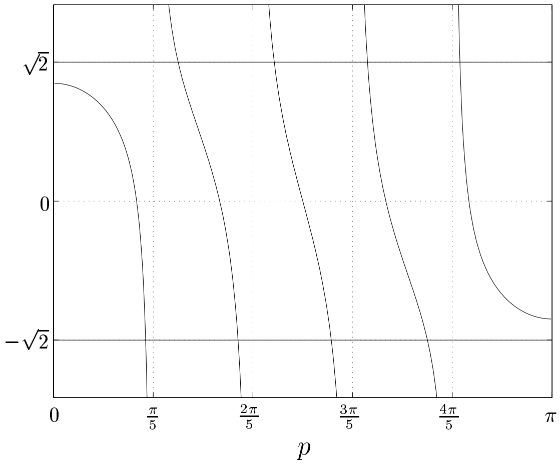

Figure 5: Left hand side of (22) for $n=5$.

Let p' and p'' be the two roots of (22) closest to the root p just found, with $p'<p<p''$. From the figure we see that p' and p'' both are roots of (22) 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

$$p' < \frac{l\pi}{n}-\frac{l\pi}{n^2}+O\!\left(\frac{1}{n^3}\right) \qquad\text{and}\qquad p''>\frac{l\pi}{n}\,, \tag{25}$$

from which we conclude that

$$p-p' > \frac{l\pi\sqrt{2}}{(1+\sqrt{2})n^2}+O\!\left(\frac{1}{n^3}\right) \quad\text{and} \tag{26}$$
$$p''-p > \frac{l\pi}{(1+\sqrt{2})n^2}+O\!\left(\frac{1}{n^3}\right), \tag{27}$$

for $l=1,2,\ldots,n-1$. 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 Δp by

$$\Delta E = 2|\Delta p\,\sin p| + O\left((\Delta p)^2\right)\,. \tag{28}$$

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

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

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 $2n-2$ roots of (22) with p real. If we let $p=ik$ with k real, we can have eigenstates of the form (14) 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 (22) becomes

$$\frac{\sinh((n+1)k)}{\sinh nk} = \pm\sqrt{2}. \tag{29}$$

As $n \to \infty$, 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 (29), and as $n \to \infty$, 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 $n \to \infty$. Since the other $n-2$ eigenvalues are all in the range $[-2, 2]$, our conclusion about the minimum spacing is unchanged. □

From Lemma 3 and Lemma 4, we conclude

THEOREM 5. *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 $\frac{1}{2n}(1-\epsilon)$.*

Combining this with the efficient implementation of the quantum walk from Section 3.2, we obtain a quantum algorithm that solves the traversal problem for G'_n using a polynomial number of calls to the black box and a polynomial number of one- and two-qubit gates.

4. CLASSICAL LOWER BOUND

In this section, we show that any classical algorithm solving the problem of traversing G'_n (as described in Section 2) requires exponential time. The precise result is

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

We shall prove Theorem 6 by considering a series of games and proving relations between them. The first game is essentially equivalent to our problem and each new game is essentially as easy to win. Finally, we will show that the easiest game cannot be won in subexponential time.

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. *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. The algorithm wins if it ever sends the oracle the name of the* EXIT *vertex.*

Note that there are two sources of randomness in this oracle: in the choice of a graph G'_n and in the random naming of its vertices. We first consider a fixed graph G 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, the success probability of the algorithm A is defined as

$$\mathbb{P}_X^G(A) = \Pr_{\text{names}}[A \text{ wins game } X \text{ on graph } G], \qquad (30)$$

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 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 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 7. *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,*

$$\mathbb{P}_1^G(A) \le \mathbb{P}_2^G(A') + O(t/2^n). \qquad (31)$$

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 $11\ldots1$. The chance of A discovering the name of a new vertex is at most $t(2^{n+2}-2)/(2^{2n}-1)$, and unless this happens, the two algorithms will have similar behavior. □

To obtain a bound on the success probability of Game 2, we will compare it with a simpler game, which is the same except that it provides an additional way to win:

GAME 3. *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 2. The algorithm wins it ever sends the oracle the name of the* EXIT *vertex, or if the subgraph it has seen contains a cycle.*

Game 3 is clearly easier to win than Game 2, so we have

LEMMA 8. *For all algorithms A for Game 2,*

$$\mathbb{P}_2^G(A) \le \mathbb{P}_3^G(A). \qquad (32)$$

Now we further restrict the form of the subgraph that can be seen by the algorithm unless it wins Game 3. 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 $\pi(u) \ne \pi(v)$ for $u \ne v$. 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 $\pi(\textsc{root}) = \textsc{entrance}$ 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 $i < j$.
2. Set $\pi(0) = \textsc{entrance}$.
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 $i = 1, 2, 3, \ldots$, 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 now define the game of finding a tree T for which a randomly chosen π is an improper embedding or T exits:

GAME 4. *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 T exits G'_n under π.*

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

$$\mathbb{P}^G(T) = \Pr_{\pi}\left[\pi \text{ is improper for } T \text{ or } T \text{ exits } G \text{ under } \pi\right],$$

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

$$\max_A \mathop{\mathrm{E}}_G \left[\mathbb{P}_4^G(A)\right] \le \max_{\text{trees } T \text{ with } t \text{ vertices}} \mathop{\mathrm{E}}_G \left[\mathbb{P}^G(T)\right]. \quad (33)$$

(Here $\mathrm{E}_G\left[\cdot\right]$ means the expectation over graphs.) Game 3 and Game 4 are also equivalent:

LEMMA 9. *For any algorithm A for Game 3 that uses at most t queries of the oracle, there exists an algorithm A' for Game 4 that outputs a tree of at most t vertices such that for all graphs G,*

$$\mathbb{P}_3^G(A) = \mathbb{P}_4^G(A'). \quad (34)$$

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 4 has the same behavior as using A in Game 3. □

Finally, we bound the probability that an algorithm wins Game 4:

LEMMA 10. *For rooted trees T of at most $2^{n/6}$ vertices,*

$$\max_T \mathop{\mathrm{E}}_G \left[\mathbb{P}^G(T)\right] \le 3 \cdot 2^{-n/6}. \quad (35)$$

PROOF. Let T be a tree with t vertices, $t \le 2^{n/6}$, with image $\pi(T)$ in G'_n under the random embedding π. The vertices of columns $n+1, n+2, \ldots n+\frac{n}{2}$ 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 $S_1, S_2, \ldots, S_{2^{n/2}}$ denote the $2^{n/2}$ subtrees described above. Let $S'_1, S'_2, \ldots, S'_{2^{n/2}}$ 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 $\le n$, as shown in Figure 6.

$\pi(P_1)$ visits a sequence of subtrees $S'_{i_1}, S_{j_1}, S'_{i_2}, \ldots$ and, similarly, $\pi(P_2)$ visits a sequence of subtrees $S'_{k_1}, S_{l_1}, S'_{k_2}, \ldots$. 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 $t \le 2^{n-1}$, we have $2^{n/2}/(2^n - t) \le 2 \cdot 2^{-n/2}$. Since there are $\binom{t}{2}$ paths P, the probability of a cycle is less than $t^2 \cdot 2^{-n/2}$.

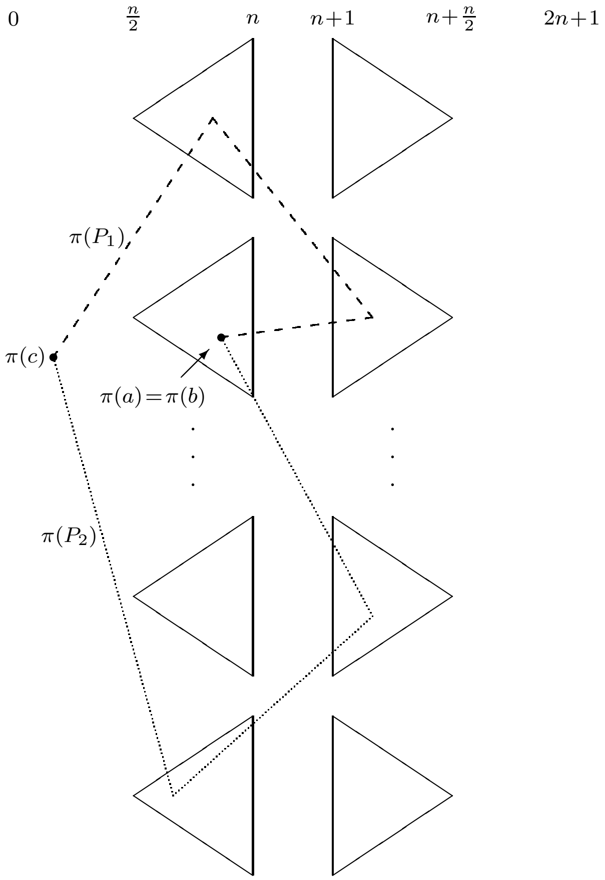

Figure 6: Graphical representation of part (ii) of the proof of Lemma 10. The triangles represent the subtrees of G'_n of height $n/2$, the dashed and dotted lines represent the paths $\pi(P_1)$ and $\pi(P_2)$, respectively, which form a cycle in the graph.

Overall we have shown that

$$\mathop{\mathrm{E}}_{G}\left[\mathbb{P}^G(T)\right] \le t^2 \cdot 2^{-n/2} + t^2 \cdot 2^{1-n/2} \le 3 \cdot 2^{-n/6}\,, \quad (36)$$

if $t \le 2^{n/6}$. □

This completes the proof of Theorem 6.

5. DISCUSSION

In this paper, we have applied a fairly general implementation of (continuous time) quantum walks to a black box problem, and 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 a black box that outputs the names of the neighbors of an input vertex. Note that although our quantum algorithm finds the name of the EXIT, it does not find a particular path from ENTRANCE to EXIT.

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 G and two vertices S and T and the goal is to determine whether or not there is a path connecting S and T. The idea is to set G to two disjoint copies of G'_n, S to the ENTRANCE of one of the copies and T to the EXIT of either the same copy or the other copy of G'_n. The quantum algorithm in Section 3 can be adapted to solve this problem in polynomial time and the lower bound of Section 4 can be adapted to show that no classical algorithm can solve this problem in subexponential time. (See [30] for a survey of classical results about graph reachability problems in various contexts.)

Our traversal problem can be viewed as a decision problem either by recasting it as a reachability problem or by asking for the first bit of the name of the EXIT. Thus, our result can be also interpreted as showing the existence of a new kind of oracle relative to which BQP ≠ BPP [6, 26].

Many computational problems can be recast as determining some property of a graph. A natural question is whether there are useful computational problems (especially non-black box 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 several helpful remarks about quantum walks.

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

6. REFERENCES

[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions* (Dover, New York, 1972).

[2] D. Aharonov, A. Ambainis, J. Kempe, and U. Vazirani. Quantum walks on graphs. In *Proc. of the 33rd ACM Symp. on Theory of Computing*, pages 50–59, 2001.

[3] D. Aharonov and A. Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. In *Proc. 35th ACM Symp. on Theory of Computing*, 2003.

[4] A. Ambainis, E. Bach, A. Nayak, A. Vishwanath, and J. Watrous. One-dimensional quantum walks. In *Proc.*

33rd ACM Symp. on Theory of Computing, pages 37–49, 2001.

[5] J. N. de Beaudrap, R. Cleve, and J. Watrous. Sharp quantum vs. classical query complexity separations. *Algorithmica*, 34:449–461, 2002.

[6] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. on Computing*, 26:1411–1473, 1997.

[7] A. M. Childs, E. Farhi, and S. Gutmann. An example of the difference between quantum and classical random walks. *Quantum Information Processing*, 1:35–43, 2002.

[8] W. van Dam and S. Hallgren. Efficient quantum algorithms for shifted quadratic character problems. quant-ph/0011067.

[9] W. van Dam and G. Seroussi. Efficient quantum algorithms for estimating Gauss sums. Unpublished.

[10] D. Deutsch. Quantum theory, the Church-Turing principle, and the universal quantum computer. *Proc. Roy. Soc. London A*, 400:96–117, 1985.

[11] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proc. Roy. Soc. London A*, 439:553–558, 1992.

[12] E. Farhi and S. Gutmann. Quantum computation and decision trees. *Phys. Rev. A*, 58:915–928, 1998.

[13] J. Goldstone. Personal communication, Sept. 2002.

[14] M. Grigni, L. Schulman, M. Vazirani, and U. Vazirani. Quantum mechanical algorithms for the nonabelian hidden subgroup problem. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 68–74, 2001.

[15] S. Hallgren. Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. In *Proc. 34th ACM Symp. on Theory of Computing*, pages 653–658, 2002.

[16] S. Hallgren, A. Russell, and A. Ta-Shma. Normal subgroup reconstruction and quantum computation using group representations. In *Proc. 32nd ACM Symp. on Theory of Computing*, pages 627–635, 2000.

[17] G. Ivanos, F. Magniez, and M. Santha. Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem. quant-ph/0102014.

[18] J. Kempe. Quantum random walks hit exponentially faster. quant-ph/0205083.

[19] A. Kitaev. Quantum measurements and the abelian stabilizer problem. quant-ph/9511026.

[20] S. Lloyd. Universal quantum simulators, Science 273:1073–1078, 1996.

[21] D. A. Meyer. From quantum cellular automata to quantum lattice gasses. *J. Stat. Phys.*, 85:551–574, 1996.

[22] C. Moore and A. Russell. Quantum walks on the hypercube. In *Proc. of RANDOM 02*, Vol. 2483 of *LNCS*, pages 164–178, 2002.

[23] M. Mosca and A. Ekert. The hidden subgroup problem and eigenvalue estimation on a quantum computer. In *Proc. 1st NASA International Conf. on Quantum Computing and Quantum Communication*, Vol. 1509 of *LNCS*, pages 174–188, 1999.

[24] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2000).

[25] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. *SIAM J. on Computing*, 26:1484–1509, 1997.

[26] D. Simon. On the power of quantum computation. *SIAM J. on Computing*, 26:1474–1483, 1997.

[27] V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Metody Diskret. Analiz.*, 3:25–30, 1964.

[28] J. Watrous. Quantum algorithms for solvable groups. In *Proc. 33rd ACM Symposium on Theory of Computing*, pages 60–67, 2001.

[29] J. Watrous. Quantum simulations of classical random walks and undirected graph connectivity. *J. Computer and System Sciences*, 62:376–391, 2001.

[30] A. Wigderson. The complexity of graph connectivity. In *Proc. 17th Mathematical Foundations of Computer Science Conf.*, Vol. 629 of *LNCS*, pages 112–132, 1992.

APPENDIX

A. EFFICIENT CLASSICAL ALGORITHM TRAVERSING THE HYPERCUBE

We describe a classical algorithm that traverses the hypercube graph, where ENTRANCE is any vertex and EXIT is 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 distance from ENTRANCE is k. In other words, the algorithm will construct a sequence of vertices $a_0 = \text{ENTRANCE}, a_1, \ldots, a_{n-1}, a_n = \text{EXIT}$, 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 $k-1$.

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 $O(n^2)$ queries, so the total number of queries is $O(n^3)$.