The quantum query complexity of determining whether a graph is connected*

Abbas Mehrabian†

Department of Combinatorics and Optimization

University of Waterloo

November 25, 2013

1 Introduction

We study the following algorithmic problem: given a graph, determine whether it is connected or not. We will call this problem *the connectivity problem* for brevity. This is a basic graph problem, and one can easily come up with an optimal algorithm based on breadth-first or depth-first search. However, we look at this problem from a different angle. First, we assume that the graph is not readily available to the algorithm, but the algorithm has a *black-box access* to the graph. Namely, the algorithm cannot access the graph directly, but can obtain information by asking questions about it. The format of these *queries* (technical term for questions) will be described later on. In most articles on algorithms, the objective is to come up with an algorithm with the shortest *running time*. Our focus is different: here we aim to solve the connectivity problem by asking as few queries as possible, hence the phrase "query complexity" in the title. Second, as also evident from the title, our focus is on *quantum algorithms* for the connectivity problem. In other words, we would like to know, quantitatively, how much advantage quantum algorithms give us over classical algorithms for this particular problem.

*This is the author's project report for the course "CO 681: Introduction to Quantum Information Processing," Fall 2013.

†email address: amehrabi@uwaterloo.ca

We allow the algorithms to use *randomness*. This is unavoidable in most quantum algorithms, as measurements of non-basis quantum states result in probabilistic outcomes. We also tolerate a small amount of error: we require the algorithm to output the correct answer with probability at least 2/3. Note that this probability involves the coin flips and (quantum) measurements of the algorithm, and not the input distribution. In other words, we require this probability of success for *every input graph*. The constant 2/3 here is somewhat arbitrary and we could have used any constant greater than 1/2, without changing the results. Also, it is easy to see that for any $\delta \in (0, 1)$, one can boost the success probability to $1 - \delta$ by repeating such an algorithm independently $O(\log(1/\delta))$ times, and outputting the most occurred answer.

We consider two models of black-box access to the input graph, the matrix model and the array model. For both models we provide quantum algorithms whose query complexities are optimal, up to constant factors. Next we describe these models and their query formats. In this paper n always denotes the number of vertices of the input graph G, and we assume that G is undirected and simple.

In the *matrix model*, the algorithm is initially given n. Assume the vertices of G are numbered from 1 to n. A query of the algorithm is of the form (i, j), where $1 \leq i < j \leq n$, whose answer is 1 if vertices i and j are joined by an edge in G, and 0 otherwise. We prove the following theorem for this model.

Theorem 1 ([2])**.** *In the matrix model, there is a quantum algorithm with query complexity $O(n\sqrt{n})$ for the connectivity problem.*

The *degree* of a vertex is the number of its incident edges. In the *array model*, the algorithm is initially given n and the degrees of all vertices. Let $d_1, d_2, \ldots, d_n$ denote the degrees of vertices $1, 2, \ldots, n$, respectively. Assume that for every $i = 1, 2, \ldots, n$, the neighbours of vertex i are ordered from 1 to d_i. This ordering is assumed to be fixed throughout the execution of the algorithm. A query of the algorithms is of the form (i, m), where $1 \leq i \leq n$ and $1 \leq m \leq d_i$, whose answer is the index of the m-th neighbour of vertex i (hence the answer is a number between 1 and n, inclusive). We prove the following theorem for this model.

Theorem 2 ([2])**.** *In the array model, there is a quantum algorithm with query complexity $O(n)$ for the connectivity problem.*

For the classical (non-quantum) case, we prove the following theorems.

model	matrix	array
classical	$\Theta(n^2)$	$\Theta(n^2)$
quantum	$\Theta(n\sqrt{n})$	$\Theta(n)$

Table 1: Query complexity of the connectivity problem in various settings.

Theorem 3 ([2])**.** *In the matrix model, there is a classical algorithm with query complexity $O(n^2)$ for the connectivity problem. Any classical algorithm for this problem in this model has query complexity $\Omega(n^2)$.*

Theorem 4. *In the array model, there is a classical algorithm with query complexity $O(n^2)$ for the connectivity problem. Any classical algorithm for this problem in this model has query complexity $\Omega(n^2)$.*

Theorems 1, 2 and 3 have been proved in [2]. We elaborate on their proofs and provide more details. It is also shown in [2] that the algorithms given by Theorems 1 and 2 are indeed tight up to constant factors. More formally, the authors have shown that any quantum algorithm for the connectivity problem has query complexity $\Omega(n\sqrt{n})$ in the matrix model, and query complexity $\Omega(n)$ in the array model. Known results on the query complexity of this problem in various settings are given in Table 1.

The authors of [2] also studied the quantum query complexity of three other problems, and provided lower and upper bounds that match up to logarithmic factors. These problems are: (1) determining whether a directed graph is strongly connected, (2) finding a minimum spanning tree of a graph, and (3) finding single source shortest paths in a directed graph.

Theorem 4 is new to the best of my knowledge and can be considered as the contribution of this article.

This article is organized as follows. Quantum algorithms for Theorems 1 and 2 are presented in Section 2. Theorems 3 and 4 are proved in Section 3.

2 Quantum algorithms

In this section we prove Theorems 1 and 2 by presenting quantum algorithms. Grover [3] presented an algorithm that given a domain of size N and black-box access to some

function $F : \{1, \ldots, N\} \to \{0, 1\}$, for which it is guaranteed that there exists some y with $F(y) = 1$, finds such a y in expected time $O(\sqrt{N})$. A *black-box access* to the function F means that the only way the algorithm can get information about F is by evaluating F on elements in the domain. The main quantum tool used in both algorithms of this section is *amplitude amplification*, which is a generalization of Grover's search algorithm. The following result, whose proof we omit, has been proved in [1] (we use the wording of [2]).

Theorem 5 (The Search Algorithm [1])**.** *Let $F : \{1, \ldots, N\} \to \{0, 1\}$ be arbitrary, $A = \{y : F(y) = 1\}$, and $a = |A|$. There is a quantum algorithm that is given black-box access to F, and does the following.*

1. *If $a > 0$, the algorithm outputs a random element in A asking an expected number of $0.9\sqrt{N/a}$ queries.*
2. *If $a = 0$, the algorithm does not halt.*

Note that the algorithm does not need to know what a is.

In this section this algorithm will be called *the search algorithm*.

2.1 Algorithm for the matrix model

Let us denote the input graph by G, and recall that n denotes the number of vertices of G. Suppose that for some function $g(n)$, we present an algorithm $\mathcal{A}$ that if G is connected, outputs YES after asking at most $g(n)$ queries on average; and does not halt if G is not connected. Consider the algorithm $\mathcal{B}$ which does the following: given input graph G, it runs $\mathcal{A}$ on G, and waits until $\mathcal{A}$ has asked $3g(n)$ queries. If $\mathcal{A}$ outputs YES by then, $\mathcal{B}$ also outputs YES, claiming that G is connected. Otherwise, i.e. if $\mathcal{A}$ has not halted by then, then $\mathcal{B}$ outputs NO, claiming that G is not connected. We claim that $\mathcal{B}$ has error probability at most $1/3$. In fact, two cases are possible:

1. If G is not connected, then $\mathcal{A}$ does not halt, hence $\mathcal{B}$ outputs NO with probability 1. The error probability is 0 in this case.
2. If G is connected, then the expected number of queries $\mathcal{A}$ asks before answering YES is at most $g(n)$. Hence by Markov's inequality, with probability at least $2/3$, $\mathcal{A}$ asks at most $3g(n)$ queries, and outputs YES before asking the $(3g(n) + 1)$-th query. If

this is the case, then $\mathcal{B}$ outputs YES. Therefore, in this case the error probability is at most 1/3.

In the following we present an algorithm that if G is connected, outputs YES after asking $O(n\sqrt{n})$ queries on average; and does not halt if G is not connected. By the discussion above, this can be used to design an algorithm that has (deterministic) query complexity $O(n\sqrt{n})$, and errs with probability at most 1/3.

By a *piece* in G, we mean a set of vertices that induce a connected subgraph of G. Let $V(G)$ denote the vertex set of G. The algorithm maintains a partition of $V(G)$ into pieces, i.e. a set $\mathcal{S}$ of disjoint pieces, such that every vertex is contained in at least one of the pieces. Given two pieces P and Q and an edge e joining a vertex of P and a vertex of Q, by *merging P and Q using e* we mean deleting pieces P and Q from $\mathcal{S}$ and adding the piece $P \cup Q$ to $\mathcal{S}$. Note that after such an operation, $\mathcal{S}$ is still a partition of $V(G)$ into pieces.

The algorithm starts with the finest possible partition, putting each vertex into a distinct piece. In each iteration, using the search algorithm it finds an edge whose endpoints lie in distinct pieces, and uses that edge to merge the corresponding pieces. If G is not connected, then the algorithm does not halt, because at some point it cannot find an appropriate edge to merge two pieces. If G is connected, then since the number of pieces is initially n and decreases by 1 in each iteration, after $n-1$ iterations all vertices will lie in the same piece. The algorithm halts and outputs YES once all pieces are merged into one.

We now analyze the query complexity of the given algorithm in the connected case. Consider an iteration in which there are k pieces, where $2 \leq k \leq n$, and the algorithm searches for an edge between distinct pieces. Since G is connected, there are at least $k-1$ such edges. Every such edge is of the form $\{i, j\}$, with $1 \leq i < j \leq n$. In the matrix model, the algorithm can ask whether $\{i, j\}$ is an edge in G using a single query. Hence the domain size is $\binom{n}{2}$. Thus the expected number of queries needed to find a desirable edge using the search algorithm is at most $0.9\sqrt{\binom{n}{2}/(k-1)}$, which is $O\left(\sqrt{n^2/k}\right)$. Since k runs between n and 2, by linearity of expectation, the expected total number of queries is

$$\sum_{k=2}^{n} O\left(\sqrt{n^2/k}\right) = O\left(n \sum_{k=2}^{n} \frac{1}{\sqrt{k}}\right) \leq O\left(n \int_{x=1}^{n} \frac{1}{\sqrt{x}}\right) = O\left(n\left[2\sqrt{n} - 2\sqrt{1}\right]\right) = O\left(n\sqrt{n}\right),$$

completing the proof of Theorem 1.

2.2 Algorithm for the array model

By the discussion in the beginning of Section 2.1, to prove Theorem 2 it is enough to present an algorithm that if G is connected, outputs YES after asking $O(n)$ queries on average; and does not halt if G is not connected.

The algorithm is similar to that of Section 2.1, in that it also maintains a partition of $V(G)$ into pieces, and in each iteration finds an edge merging two distinct pieces. However, to get a linear average query complexity, our approach should be more delicate, both in selecting the initial set of pieces, and in finding an appropriate edge for merging.

We define the *total degree* of a piece P, written $t(P)$, as the sum of degrees of vertices in P. The following result, whose proof will be presented later on, will be used for building the initial piece set.

Lemma 6 ([2])**.** *There is a deterministic classical algorithm that asks $O(n)$ queries and partitions $V(G)$ into several pieces in such a way that for each piece P, its total degree is less than $|P|^2$.*

The algorithm has two phases. In Phase (1), we use Lemma 6 to partition $V(G)$ into pieces asking $O(n)$ queries. In Phase (2), we merge the pieces iteratively: in each iteration, we choose a piece P with minimum total degree, use the search algorithm to find an edge with one endpoint in P and the other one outside P, and then merge two pieces using this edge. We stop and output YES once all pieces are merged into one.

It is clear that if G is not connected, then the algorithm will not halt. The following lemma implies that if G is connected, then in Phase (2) the algorithm asks $O(n)$ queries on average, thus proving Theorem 2. The proof presented here is new, and simpler than the original proof in [2].

Lemma 7 ([2])**.** *Denote by $\{P_1, P_2, \ldots, P_k\}$ the partition of $V(G)$ into pieces after Phase (1). If G is connected, then the expected query complexity of Phase (2) is*

$$O\left(\sqrt{t(P_1)} + \sqrt{t(P_2)} + \cdots + \sqrt{t(P_k)}\right) \leq O(n)\,.$$

Proof. The inequality $\sqrt{t(P_1)} + \sqrt{t(P_2)} + \cdots + \sqrt{t(P_k)} < n$ holds since by Lemma 6, $\sqrt{t(P_j)} < |P_j|$ for each initial piece P_j, and moreover, the pieces form a partition of $V(G)$.

Assume the input graph G is connected. Using induction on k, we prove that the

expected query complexity of Phase (2) is not more than $2\sum_{s=1}^{k}\sqrt{t(P_s)}$. This is clear if $k = 1$, so let us assume $k > 1$.

Assume that P_i is a piece with minimum total degree, where $1 \le i \le k$. In the first iteration, we search for an edge with exactly one endpoint in P_i. Recall that for a vertex j, d_j denotes the degree of j. Hence we are indeed searching for a pair (j, m) with $j \in P_i$, $1 \le m \le d_j$, such that the m-th neighbour of vertex j is not in P_i. In the array model, the m-th neighbour of vertex j can be determined by asking a single query. Hence the domain size is $\sum_{j\in P_i} d_j = t(P_i)$, and there is at least one desirable edge since the graph is connected. Hence the first iteration is completed after asking an average of at most $0.9\sqrt{t(P_i)}$ queries.

Suppose that in the first iteration pieces P_i and P_r are merged, where $r \in \{1, \ldots, n\} \setminus \{i\}$. Then by the inductive hypothesis, the expected query complexity of the remaining iterations is at most

$$2\left(\sum_{\substack{1\le s\le k\\ s\ne i,r}}\sqrt{t(P_s)} + \sqrt{t(P_i)+t(P_r)}\right).$$

By the linearity of expectation, to prove the inductive step we just need to show

$$0.9\sqrt{t(P_i)} + 2\left(\sum_{\substack{1\le s\le k\\ s\ne i,r}}\sqrt{t(P_s)} + \sqrt{t(P_i)+t(P_r)}\right) \le 2\sum_{s=1}^{k}\sqrt{t(P_s)}\,. \tag{1}$$

Cancelling the common terms, this is equivalent to

$$2\sqrt{t(P_i)+t(P_r)} \le 1.1\sqrt{t(P_i)} + 2\sqrt{t(P_r)}\,.$$

Squaring both sides, this is equivalent to

$$4t(P_i) + 4t(P_r) \le 1.21t(P_i) + 4t(P_r) + 4.4\sqrt{t(P_i)t(P_r)}\,,$$

which follows from $t(P_i) \le t(P_r)$. Hence (1) is proved, and this completes the proof of the inductive step and that of the lemma. ■

Finally, we prove Lemma 6.

Proof of Lemma 6. Algorithm 1 shown on the next page does the job. Initially, there is no piece and all vertices are *uncovered*, i.e. not contained in any piece. In every iteration,

the algorithm either builds a new piece, or adds a group of vertices to an existing piece. The algorithm repeats this until all vertices are covered.

Algorithm 1 Partition $V(G)$ into pieces with small total degrees

1: Start with no piece and all vertices uncovered.
2: **while** there exist uncovered vertices **do**
3: Let u be an uncovered vertex with maximum degree.
4: Go through neighbours of u one by one, inserting them into a buffer B:
5: **if** a covered neighbour v is encountered **then**
6: add $\{u\} \cup B$ to the piece containing v and mark them as covered; jump to Line 2.
7: **else if** all neighbours of u are uncovered **then**
8: Let u and its neighbours form a new piece and mark them as covered; jump to Line 2.
9: **end if**
10: **end while**

It is clear that Algorithm 1 is deterministic and classical, and that it outputs a partition of $V(G)$ into pieces. We claim that its query complexity is not more than $2n$. Line 4 is the only place in Algorithm 1 where queries are asked. Every vertex, while it is uncovered, is the answer of at most one query, since in that very iteration it will be put into some piece. The algorithm has n iterations at most, and in each iteration at most one covered vertex is the answer of a query, since as soon as such a vertex is encountered (Line 5), a group of vertices are covered and the iteration finishes. Hence the total query complexity of the algorithm is not more than $2n$, which is $O(n)$.

It remains to show that when Algorithm 1 is done, all generated pieces have small total degrees. We show that, indeed, throughout the execution of the algorithm each piece P has total degree less than $|P|^2$. Changes in pieces occur in two places: in Line 6, where a group of vertices is added to an already-present piece, and in Line 8, where a new piece is created. We study these cases separately, starting with the latter case. Consider an iteration of Algorithm 1. Let u be as in the algorithm, and let d denote the degree of u.

Case 1. In Line 8 of Algorithm 1, a new piece is created, containing u and its neighbours. The new piece contains $d + 1$ vertices. By the choice of u in Line 3, each neighbour of u has degree at most d, and u has exactly d neighbours, so the total degree of the new piece is not more than $d + d \times d$, which is less than $(d + 1)^2$. Observe that when this piece is created, every uncovered vertex has degree at most d. During the algorithm the size of a

piece never decreases. Hence the following claim holds.

Claim. Throughout the execution of Algorithm 1, the size of every piece is greater than the degree of every uncovered vertex.

Case 2. In Line 6 of Algorithm 1, vertices in $\{u\} \cup B$ are added to the piece containing v. We assume $v \notin B$. Let P denote the piece containing v, and let $b = |B|$. Before the start of this iteration u was uncovered, so by the claim above we have $|P| > d$. Also, by the choice of u in Line 3, each vertex in B has degree at most d, so $t(B) \leq bd$. Finally, piece P has already been created, so $t(P) < |P|^2$. Consequently, we find that the total degree of the new piece $P \cup \{u\} \cup B$ is

$$\begin{aligned} t(P \cup \{u\} \cup B) &= t(P) + d + t(B) \\ &\leq t(P) + d + db \\ &= t(P) + d(1+b) \\ &< |P|^2 + d(1+b) \\ &< |P|^2 + |P|(1+b) \\ &< (|P| + 1 + b)^2 \\ &= |P \cup \{u\} \cup B|^2 , \end{aligned}$$

and this concludes the proof of the lemma. ■

3 Classical lower bounds

In this section we prove Theorems 3 and 4. Given a graph G with e edges, it is easy to determine whether G is connected by performing a depth-first search on G. The time complexity of this algorithm is $O(n^2)$ in the matrix model and $O(e)$ in the array model. Since asking every query takes a unit time step, the time complexity is an upper bound for the query complexity, and the upper bound in Theorem 3 follows. The upper bound in Theorem 4 follows from the fact that every graph has at most $n(n-1)/2 = O(n^2)$ edges.

Proof of the lower bound in Theorem 3. Assume that n is even, say $n = 2m$, and define two graphs G_1 and G_2 as follows. The graph G_1 has all edges of the form $\{k, k+1\}$ for $k = 1, 2, \ldots, m-2, m-1, m+1, m+2, \ldots, n-1$. In graph theoretic terms, The graph G_1 consists of two paths, one passing through vertices $1, 2, \ldots, m$ in this order, and the

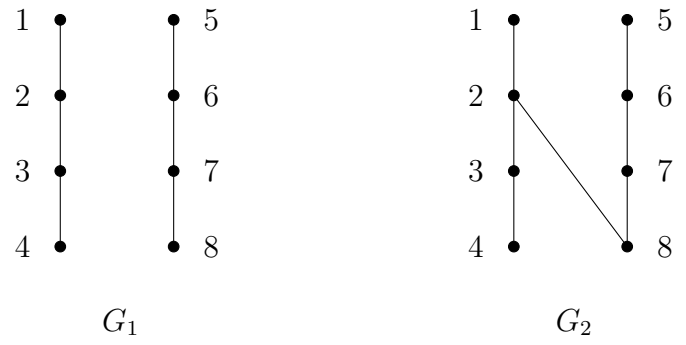
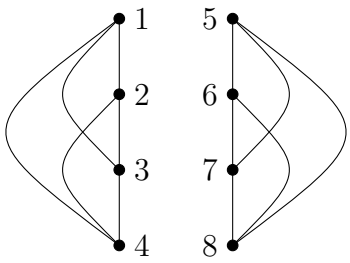


Figure 1: Illustration of the construction in the proof of the lower bound of Theorem 3.

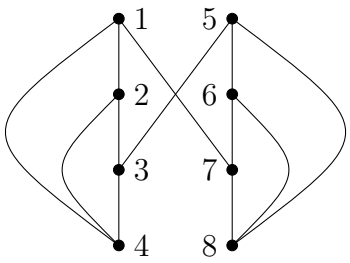
other one passing through vertices $m+1, m+2, \ldots, n$ in this order. Clearly G_1 is not connected. The graph G_2 is obtained from G_1 by adding an edge joining a random vertex of $\{1, 2, \ldots, m\}$ and a random vertex of $\{m+1, \ldots, n\}$. Clearly G_2 is connected. An illustration with $n = 8$ is depicted in Figure 1.

We claim that any classical algorithm that distinguishes between G_1 and G_2 with probability at least $2/3$ has query complexity $\Omega(n^2)$ in the matrix model. Let $A = \{1, \ldots, m\}$, $B = \{m+1, \ldots, n\}$, and let $\{a, b\}$ denote the extra edge of G_2, with $a \in A$ and $b \in B$ (we have $a = 2$ and $b = 8$ in Figure 1). Then there is exactly one query whose answer is different in the two graphs, namely (a, b). The algorithm must ask this query to distinguish between G_1 and G_2 with probability more than a half (i.e. better than a random guess). Let us call the pair (a, b) the *identifying pair*. Since a is randomly chosen in A and b is randomly chosen in B, any pair in $A \times B$ is equally likely to be the identifying pair. Thus any classical algorithm needs to ask $\Omega(|A \times B|) = \Omega(n^2)$ queries to find the identifying pair with (at least) constant probability and distinguish between G_1 and G_2 with probability more than a half. ■

Proof of the lower bound in Theorem 4. Assume that n is even and greater than 4, and let $m = n/2$. Define two graphs H_1 and H_2 as follows. The graph H_1 has all edges of the form $\{i, j\}$ for $1 \le i < j \le m$, and all edges of the form $\{i, j\}$ for $m+1 \le i < j \le n$, and has no other edges. In graph theoretic terms, H_1 consists of two complete graphs, one on vertices $\{1, \ldots, m\}$ and the other one on vertices $\{m+1, \ldots, n\}$. Clearly H_1 is not connected. The graph H_2 is obtained from H_1 as follows. Choose a pair (a, b) with $1 \le a < b \le m$ uniformly at random, delete the edges $\{a, b\}$ and $\{m+a, m+b\}$, and then add the edges $\{a, b+m\}$ and $\{b, a+m\}$. An illustration with $n = 8$, $(a, b) = (1, 3)$ is depicted in Figure 2. Clearly H_2 is connected.



H_1



H_2

vertex	neighbours		
1	2	4	***3***
2	1	3	4
3	4	2	***1***
4	3	1	2
5	***7***	8	6
6	5	7	8
7	8	***5***	6
8	7	6	5

M_1

vertex	neighbours		
1	2	4	***7***
2	1	3	4
3	4	2	***5***
4	3	1	2
5	***3***	8	6
6	5	7	8
7	8	***1***	6
8	7	6	5

M_2

Figure 2: Illustration of the construction in the proof of the lower bound of Theorem 4.

We claim that any classical algorithm that distinguishes between H_1 and H_2 with probability at least $2/3$ has query complexity $\Omega(n^2)$ in the array model. For $i = 1, 2$, define an $n \times (m-1)$ matrix M_i corresponding to H_i as follows. The j-th row of M_i is a list of neighbours of vertex j. See Figure 2 for an example. Notice that a classical algorithm with every query in fact reads the value of the corresponding matrix at a particular entry.

Suppose a classical algorithm is given the graph H_p for an unknown $p \in \{1, 2\}$, and must determine whether p is 1 or 2. Note that the matrices M_1 and M_2 differ in exactly four entries. These entries are bolded in Figure 2. The algorithm must ask the value of M_p at one or more of these special entries to distinguish between H_1 and H_2 with probability more than a half (i.e. better than a random guess). Although these four entries are not independently chosen, the crucial observation is that each of the two "top" special entries is chosen uniformly at random from the "top-half" of M_p. Similarly, each of the two bottom special entries is chosen uniformly at random from the bottom-half of M_p. The number of entries of M_p is $n \times (m-1) = \Omega(n^2)$, so the size of each half is $\Omega(n^2)$ as well. Thus any classical algorithm needs to ask $\Omega(n^2)$ queries to find a special entry with (at least) constant probability and distinguish between H_1 and H_2 with probability more than a half. ■

References

[1] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998.

[2] Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM J. Comput.*, 35(6):1310–1328, 2006.

[3] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996)*, pages 212–219, New York, 1996. ACM.