# Optimal Algorithms for Finding Connected Components of an Unknown Graph

Weiping Shi[1] and Douglas B. West[2]

[1] Department of Computer Science
University of North Texas, Denton, TX 76203, USA
[2] Department of Mathematics
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA

**Abstract.** We want to find the connected components of an unknown graph $G$ with a known vertex set $V$. We learn about $G$ by sending an oracle a query set $S \subseteq V$, and the oracle tells us the vertices connected to $S$. We want to use the minimum number of queries, adaptively, to find the components. The problem is also known as interconnect diagnosis of wiring networks in VLSI. The graph has $n$ vertices and $k$ components, but $k$ is not part of the input. We present a deterministic algorithm using $O(\min\{k, \log n\})$ queries and a randomized algorithm using expected $O(\min\{k, \log k + \log\log n\})$ queries. We also prove matching lower bounds.

## 1 Introduction

In this paper, we study how to find connected components of an unknown undirected graph $G = (V, E)$. Vertices $u$ and $v$ are *connected* if there is a path between them. The connection relation is an equivalence relation on the vertex set $V$. The *components* of $G$ are its maximal connected subgraphs, and the vertex sets of the components are the equivalence classes of the connection relation. In our search problem, we are given $V$ but not the set $E$ of edges. Also we do not know the number of components or their sizes. The only operation we may use to obtain information about $G$ is to query an oracle. For any query set $S \subseteq V$, the oracle will tell us $Q(S)$, the set of vertices connected to vertices of $S$:

$$Q(S) = \bigcup_{u \in S} \{v \in V \mid \ u \text{ and } v \text{ are connected.}\}$$

Our objective is to find the components, adaptively, using the minimum number of queries.

This problem comes from the interconnect diagnosis of wiring networks of logic circuits. It has applications to design and testing of very large scale integration (VLSI), multi-chip module (MCM) and printed circuit board (PCB) systems [2, 5, 7, 9]. A wiring network consists of a set of nets, each having one driver and one receiver. The logic value of a good net is specified by its driver and observed by its receiver. When some nets are involved in a short fault, their

receivers all receive the logical OR of the values of their drivers. To diagnose a wiring network, a test engineer sends a "test vector" of logical 0's and 1's from the drivers and observes the outputs. The task of interconnect diagnosis is to adaptively apply test vectors to the nets to identify all the faults. Diagnosing a wiring network is the same as finding the connected components of the graph of short faults; applying test vectors is the same as querying the oracle.

Kautz [7] studied the problem for the special case of testing $G = \overline{K}_n$. Garey, Johnson and So [4] observed that with the complication of partial information about $G$, minimizing the number of queries to test $G = \overline{K}_n$ is NP-complete (reduction from chromatic number). For our problem of identifying all components, Jarwala and Yau [5] provided an approach using $\log n + (n - k)$ queries. Cheng et al. [2] studied non-adaptive versions of the problem, where the inputs of all queries are decided before asking the oracle any question. Shi and Fuchs [9] proved $n-1$ queries are necessary and sufficient to find all components nonadaptively, They also presented a recursive version of the deterministic algorithm of Section 2, for the special case of the interconnect diagnosis problem. Chen and Hwang [1] used a different model, called "group testing", where the inputs of each query are two sets $S$ and $T$, and the oracle answers "yes" or "no" depending on whether some vertex in $S$ is connected to some vertex in $T$.

Table 1 summaries our results, where $n$ is the number of vertices and $k$ is the number of components, which is not part of the input. We measure the query complexity in terms of both the input size $n$ and the output size $k$. We make no assumption on $k$ other than $1 \leq k \leq n$. We present algorithms achieving the upper bounds and prove matching lower bounds. Note that randomization may permit an exponential reduction in the number of queries in terms of the input size. This effect was first observed by Fiat et al. in studying layered graph traversal [3]. Recently, Kavraki et al., [8] also studied the problem of finding connected components of an unknown graph, but their oracle takes only one vertex in each query.

**Table 1.** Number of queries required to find connected components.

| | |
|---|---|
| **Deterministic** | $\Theta(\min\{k, \log n\})$ |
| **Randomized** | $\Theta(\min\{k, \log k + \log\log n\})$ |
| **Nondeterministic** | $\Theta(\log k)$ |

## 2 Deterministic Algorithm

There is a straightforward deterministic algorithm [5] to find the components in $k$ queries: Iteratively pick a vertex $v$, use it as a query, record $Q(\{v\})$ as a component, and delete $Q(\{v\})$. Each query finds one new component. The

upper bound of $k$ can be improved when $n < 2^k$; we present a divide-and-conquer algorithm that uses $\log n$ queries to find all components. The algorithm iteratively maintains a *component structure* $P = \{(S_j, R_j) \mid j = 1, 2, \ldots, t\}$, where $\{S_j\}$ is a collection of subsets of vertices that form a partition of $V$, and $\{R_j\}$ is a collection of "chosen" subsets of vertices such that $R_j \subseteq S_j$ and $Q(R_j) = S_j$. This property of the chosen subsets implies that each $S_j$ is a union of components. The aim is to refine the partition into components and to reduce the chosen subsets to single vertices; the algorithm terminates when each $R_j$ is a single vertex in a component $S_j$.

**Algorithm** 1.
**Input:** A set of vertices $V$, $|V| = n$.
**Output:** All connected components.
**Method:**
1: $P \leftarrow \{(V, V)\}$.
2: **For** $i \leftarrow 0$ **to** $\log n$
3: $\quad P \leftarrow \mathcal{D}(P)$.
4: **For** each $(S_j, R_j) \in P$
5: $\quad$ Report $S_j$ as one component.
**End of Algorithm.**

**Procedure** $\mathcal{D}(P)$.
**Input:** A component structure $P = \{(S_j, R_j) \mid j = 1, 2, \ldots, t\}$.
**Output:** A refined component structure $P' = \{(S'_j, R'_j) \mid j = 1, 2, \ldots, t'\}$.
**Method:**
1: **For** each $j$ such that $|R_j| > 1$,
2: $\quad$ Arbitrarily pick $R'_j \subset R_j$ such that $|R'_j| = \lceil |R_j|/2 \rceil$.
3: Perform the single query $\cup R'_j$, with result $S' \leftarrow Q(\cup R'_j)$.
4: $P' \leftarrow \emptyset$.
5: **For** each $j$ such that $|R_j| > 1$,
6: $\quad$ Let $T_j = S_j \cap S'$.
7: $\quad P' \leftarrow P' \cup \{(T_j, R'_j)\}$.
8: $\quad$ **If** $T_j \neq S_j$ **then** $P' \leftarrow P' \cup \{(S_j - T_j, R_j - T_j)\}$.
9: **Return** $P'$.
**End of Procedure.**

For example, suppose $G$ has two components $G_1$ and $G_2$. Suppose in the first call of Procedure $\mathcal{D}(P)$, $P = \{(V, V)\}$, $R'_1$ has vertices from both components. Then $S' = V$, and $P' = \{(V, R'_1)\}$. In other words, the new partition still has all vertices in a single block, but the chosen subset is smaller. Applying the procedure again and suppose this time $R'_1$ has vertices only from $G_1$. Then $T_1$ will be $V(G_1)$ and $S_1 - T_1$ will be $V(G_2)$. The corresponding chosen subsets for $S_1 - T_1$ will contain vertices only in $G_2$. Maintaining the partition that we have discovered enables us to disassemble the information in steps 5-8 from the single combined query in step 3.

**Theorem 1.** *Given a component structure in which the largest chosen subset has $m$ vertices, $\log m$ iterations of Procedure $\mathcal{D}$ (one query each) finds the components of the graph as the blocks of the final component structure.*

*Proof.* With each query, we divide by 2 the maximum size of the chosen subsets. Hence there are at most $\log m$ queries. To prove that the algorithm works it suffices to show that the property $Q(R_j) = S_j$ is maintained by Procedure $\mathcal{D}$. If so, then when all $|R_j| = 1$, all vertices in $S_j$ are connected to the chosen vertex, and $S_j$ is the component containing it.

When some $|R_j| > 1$, the procedure performs a query and splits $S_j$ into $T_j$ and $S_j - T_j$. Because $Q(R_j) = S_j$, the query $Q(\cup R'_j)$ tells us that $Q(R'_j) = T_j$. By definition of connection, there cannot be any path between $T_j$ and $S_j - T_j$. If $S_j - T_j$ is nonempty, then it has additional components. Since all of $S_j$ was connected to $R_j$, we now have $Q(R_j - T_j) = S_j - T_j$. □

Note that the number of queries depends only on the size of the largest chosen subset; we will use this property in the randomized algorithm. Note also that an algorithm using $2(\min\{k, \log n\})$ queries can be obtained by alternating between the $k$-query algorithm and Algorithm 1.

## 3 Randomized Algorithm

In this section, we present a randomized algorithm that can sometimes reduce the number of queries exponentially. The algorithm first calls the randomized Procedure $\mathcal{R}$ for $\log\log n$ iterations, and then it calls the deterministic Procedure $\mathcal{D}$ to complete the refinement of the partition.

**Algorithm** 2.
**Input:** A set of vertices $V$, $|V| = n$.
**Output:** All connected components.
**Method:**
1: $P \leftarrow \{(V, V)\}$.
2: **For** $i \leftarrow 0$ **to** $\log\log n$ **do**
3: $\quad P \leftarrow \mathcal{R}(P)$.
4: **While** there exists $(S_j, R_j) \in P$ such that $|R_j| > 1$
5: $\quad P \leftarrow \mathcal{D}(P)$.
6: **For** each $(S_j, R_j) \in P$
7: $\quad$ Report $S_j$ as one component.
**End of Algorithm.**

Now we describe Procedure $\mathcal{R}$. Again we maintain a partition and chosen subsets; the difference is the rapid reduction in the size of the query set.

**Procedure** $\mathcal{R}(P)$.
**Input:** A component structure $P = \{(S_j, R_j) \mid j = 1, 2, \ldots, t\}$.
**Output:** A refined component structure $P' = \{(S'_j, R'_j) \mid j = 1, 2, \ldots, t'\}$.

**Method:**

1: **For** each $j$ such that $|R_j| > 1$,
2: Randomly pick $R'_j \subset R_j$ such that $|R'_j| = \left\lceil \sqrt{|R_j|} \right\rceil$.
3: Perform the single query $\cup R'_j$, with result $S'$.
4: $P' \leftarrow \emptyset$.
5: **For** each $j$ such that $|R_j| > 1$,
6: Let $T_j = S_j \cap S'$.
7: $P' \leftarrow P' \cup \{(T_j, R'_j)\}$.
8: **If** $T_j \neq S_j$ **then** $P' \leftarrow P' \cup \{(S_j - T_j, R_j - T_j)\}$.
9: **Return** $P'$.

**End of Procedure.**

The only difference between Procedure $\mathcal{R}$ and Procedure $\mathcal{D}$ is in line 2, where we randomly pick $\sqrt{|R|}$ vertices instead of $|R|/2$ vertices. The idea is still to cut $S_j$ into $S'_j$ connected to $R'_j$ and $S_j - S'$ connected to $R_j - S'$. We will prove that after $\log\log n$ iterations of $\mathcal{R}$, the maximum size of the chosen subsets will be at most $k^4 \log n$ with probability at least $1 - 1/\log n$. Therefore $4\log k + \log\log n$ iterations of Procedure $\mathcal{D}$ will complete the search.

The correctness of Algorithm 2 follows as in Theorem 1, but the complexity analysis is more delicate. Let $T(r, i)$ be the maximum of the expected number of queries used after iteration $i$, where the maximum is taken over all graphs and all component structures on $n$ vertices in which $r$ is the maximum size of a chosen subset and $i$ iterations have been performed. We have the following probabilistic recurrence relation:

$$T(r,i) \leq \begin{cases} 0 & \text{if } r = 1, \\ \log r & \text{if } r > 1,\ i = \log\log n, \\ 1 + \max\{T(\sqrt{r}, i+1), T(h(r), i+1)\} & \text{if } r > 1,\ i < \log\log n. \end{cases}$$

where $h()$ is a random variable representing the maximum number of vertices in the chosen subsets of the output $P'$. In the rest of this section, we will use martingales to estimate $E[T(n,0)]$. Karp [6] developed some general methods that apply to solutions of order $\Omega(\log n)$.

**Lemma 2.** *If $0 \leq \alpha \leq 1$ and $m$ is a positive integer, then*

$$\alpha(1-\alpha)^m < \frac{1}{em}.$$

*Proof.* The derivative of $g(\alpha) = \alpha(1-\alpha)^m$ is

$$\frac{dg(\alpha)}{d\alpha} = (1-\alpha)^m - m\alpha(1-\alpha)^{m-1} = (1-\alpha)^{m-1}(1-\alpha-m\alpha).$$

If $g'(\alpha) = 0$, then one of the factors $(1-\alpha)$ and $(1-\alpha-m\alpha)$ must be zero, which happens only at $\alpha = 1$ or $\alpha = \frac{1}{m+1}$. Checking the values of $g$ at these

points and the boundary $\alpha = 0$, we find that $g(\alpha)$ reaches its maximum when $\alpha = \frac{1}{m+1}$. Therefore,

$$g(\alpha) \leq \frac{1}{m+1}\left(1 - \frac{1}{m+1}\right)^m < \frac{1}{m+1}\frac{1}{e} < \frac{1}{em}.$$

□

**Lemma 3.** *If $f$ is a concave function and $X$ is an integer-valued random variable, then $E[f(X)] \leq f(E[X])$.*

*Proof.*

$$E[f(X)] = \sum_i \Pr[X = i] \cdot f(i) \leq f\left(\sum_i \Pr[X = i] \cdot i\right) = f(E[X]).$$

□

Given $n$ and $k$, define a sequence of random variables $X_0, X_1, \ldots, X_{\log\log n}$ by setting $X_0 = n$ and $X_i = \max\{\sqrt{X_{i-1}}, h(X_{i-1})\}$ for $i > 0$. Intuitively, $X_i$ is a bound on the size of the maximum chosen subset after $i$ iterations.

**Lemma 4.** *For the sequence of random variables defined above, the conditional expectation $E[X_i \mid X_{i-1}]$ is bounded by*

$$E[X_i \mid X_{i-1}] < \sqrt{X_{i-1}} \cdot k^2.$$

*Proof.*

$$\begin{aligned} E[X_i \mid X_{i-1}] &= E[\max\{\sqrt{X_{i-1}}, h(X_{i-1})\} \mid X_{i-1}] \\ &\leq E[\sqrt{X_{i-1}} + h(X_{i-1}) \mid X_{i-1}] \\ &= E[\sqrt{X_{i-1}} \mid X_{i-1}] + E[h(X_{i-1}) \mid X_{i-1}] \\ &= \sqrt{X_{i-1}} + E[h(X_{i-1})|X_{i-1}]. \end{aligned}$$

Now we examine $h(X_{i-1})$. Let $P = \{(S_j, R_j) \mid j = 1, 2, \ldots, t\}$ be an arbitrary component structure in which the largest of the chosen subsets is $R_j$ of size $X_{i-1}$. Suppose that $G$ has components $G_1, \ldots, G_k$, and their contributions to $R_j$ have sizes $n_1, \ldots, n_k$, with $n_l = |V(G_l) \cap R_j|$. When we apply $\mathcal{R}(P)$, The chosen subset $R_j$ is split into a chosen subset of size $m = \sqrt{X_{i-1}}$, and possibly a new block $S_j - S'$ has chosen subset $R'$ consisting of the vertices of $R_j$ not in $Q(R'_j)$. These leftover vertices belong to the components not hit by the vertices of $R'_j$. Each time we select a vertex for $R'_j$, it has probability $n_l/|R_j|$ of belonging to $G_l$. Hence the probability of missing $G_l$ completely is $(1 - n_l/|R_j|)^m$. When we do miss $G_l$, it contributes $n_l$ vertices to $R'_j$. Hence the expected size of the leftover set is $\sum_{l=1}^{k} n_l(1 - n_l/|R_j|)^m$. This expression grows with $|R_j|$. Furthermore, there are never more than $k$ blocks in the partition. We conclude that

$$
\begin{aligned}
E[h(X_{i-1}) \mid X_{i-1}] &= E\left[\max_{1\le j\le t} |R_j - T_j| \mid X_{i-1}\right] \\
&\le E\left[\sum_{j=1}^{t} |R_j - T_j| \mid X_{i-1}\right] \\
&\le k \max_{1\le j\le t} E[|R_j - T_j| \mid X_{i-1}] \\
&= k\sum_{l=1}^{k} n_l \left(1 - \frac{n_l}{X_{i-1}}\right)^m \\
&= kX_{i-1}\sum_{l=1}^{k} \frac{n_l}{X_{i-1}} \left(1 - \frac{n_l}{X_{i-1}}\right)^m .
\end{aligned}
$$

Letting $\alpha_l = n_l/X_{i-1}$, we have $\sum_{l=1}^{k} \alpha_l = 1$ and $0 \le \alpha_l \le 1$ for $l = 1, 2, \ldots, k$. Using Lemma 2 and $m = \sqrt{X_{i-1}}$, we obtain

$$
\begin{aligned}
E[h(X_{i-1}) \mid X_{i-1}] &\le kX_{i-1}\sum_{l=1}^{k} \alpha_l(1-\alpha_l)^m \\
&\le kX_{i-1}k \max_{0\le\alpha\le 1} \alpha(1-\alpha)^m \\
&< \frac{k^2 X_{i-1}}{em} = \frac{k^2\sqrt{X_{i-1}}}{e}.
\end{aligned}
$$

For $k \ge 2$, we conclude that

$$
E[X_i \mid X_{i-1}] < \sqrt{X_{i-1}} + \frac{k^2\sqrt{X_{i-1}}}{e} < \sqrt{X_{i-1}} \cdot k^2.
$$

□

**Theorem 5.** *For graphs with $n$ vertices and $k$ components, the expected number of queries used by Algorithm 2 to find all components is $O(\log k + \log\log n)$.*

*Proof.* Consider the maximum size of the chosen subsets after $i$ iterations. From Lemma 3 and Lemma 4, we have

$$
E[X_i] < E[\sqrt{X_{i-1}} \cdot k^2] \le \sqrt{E[X_{i-1}]} \cdot k^2 < \cdots < E[X_0]^{\frac{1}{2^i}} k^4 = n^{\frac{1}{2^i}} k^4.
$$

With $i = \log\log n$, we obtain $E[X_{\log\log n}] < k^4$. From Markov's inequality,

$$
\Pr[X_{\log\log n} \ge k^4 \log n] < \frac{k^4}{k^4 \log n} = \frac{1}{\log n}.
$$

With probability at most $1/\log n$, we may be left with huge chosen subsets after $\log\log n$ iterations; Procedure $\mathcal{D}$ can resolve these instances with at most $\log n$

further iterations. Since we always call Procedure $\mathcal{D}$ after $\log\log n$ iterations of Procedure $\mathcal{R}$, we have

$$E[T(n,0)] \leq \log\log n + \log(k^4 \log n) + \frac{1}{\log n}\log n \leq 2\log\log n + 4\log k + 1.$$

□

## 4 Lower Bounds

**Theorem 6.** *On a graph with $k$ components, every algorithm to find the components uses at least $\log k$ queries.*

*Proof.* Since the response to a query $Q(S)$ cuts each subset $U \subseteq V$ that is known to be a union of components into at most two disconnected subsets $U \cap Q(S)$ and $U - Q(S)$, we need at least $\log k$ queries to separate the set of vertices into $k$ components. □

Theorem 6 implies a lower bound for any nondeterministic algorithm. On the other hand, a nondeterministic algorithm can guess the components and use $\lceil \log k \rceil + 1$ queries to verify them.

**Theorem 7.** *On an $n$-vertex graph with $k$ components, every deterministic algorithm uses at least $\min\{k, \log n\}$ queries.*

*Proof.* Consider the following adversary. If we pick a set $S$ of vertices to make the first query, then our adversary makes $S$ one component if $|S| \leq n/2$, or makes $V - S$ one component otherwise. Therefore after the first query, we are left with a sub-problem of $k-1$ components and at least $n/2$ vertices. □

The randomized lower bound is more difficult to derive. We will use Yao's corollary [10] of von Neumann's minimax principle. Consider a matrix game in which the strategies of the row player are deterministic algorithms, the strategies of the column player are input instances, and the payoff is the cost of the deterministic algorithm on the input instance. The row player seeks to minimize cost by randomizing over deterministic algorithms; this is a randomized algorithm. The column player is the adversary. Yao applied the von Neumann minimax theorem to this game, obtaining the result that the expected performance of the "optimal" randomized algorithm on the worst input instance for it equals the expected cost of the worst input distribution against the best deterministic algorithm for it. More precisely, if $P$ denotes a distribution over deterministic algorithms $A$ and $Q$ denotes a distribution over input instances $G$, we have

$$\min_P\{\max_G E_P(c(A,G))\} = \max_Q\{\min_A E_Q(c(A,G))\}.$$

Hence to provide a lower bound for a randomized algorithm, it suffices to prove a lower bound for the expectation of every deterministic algorithm against a particularly bad input distribution.

We will apply this to $n$-vertex graphs generated at random as follows. There are $k$ components $C_1, \ldots, C_k$ that are cliques of sizes $n^{\epsilon_1}, n^{\epsilon_2}, \ldots, n^{\epsilon_k}$, respectively, except for a slight adjustment. We define $\epsilon_1 = 0$, $\epsilon_2 = 1$ and $\epsilon_3 = 1/2$. For $i \geq 4$, $\epsilon_i$ is chosen with the following distribution:

$$\Pr\left[\epsilon_i = \epsilon_{i-1} + \left(\frac{1}{2}\right)^{i-2}\right] = \Pr\left[\epsilon_i = \epsilon_{i-1} - \left(\frac{1}{2}\right)^{i-2}\right] = \frac{1}{2}. \tag{1}$$

The sizes of the first three cliques are $1, n$ and $\sqrt{n}$. The total number of vertices generated is $\sum_{i=1}^{k} n^{\epsilon_i} = n + o(n)$. To obtain an $n$-vertex graph, we delete the excess $o(n)$ vertices from the giant clique that initially has order $n$. We are left with an isolated vertex, a giant component, and $k-2$ intermediate components. The assignment of vertices to components is made randomly.

Fig. 1 is an example distribution of the exponents. Since it becomes crowded quickly, we only provide labels up to $\epsilon_7$.

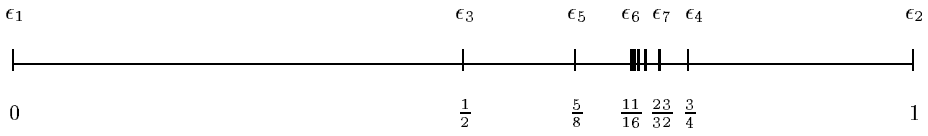


**Fig. 1.** Example distribution of the exponents.

**Lemma 8.** *Given positive real numbers $x_1, \ldots, x_k$ and an integer $n$, let $P_1 = P(x_1, \ldots, x_k)$ be the distribution over graphs consisting of disjoint cliques of orders $n^{x_1}, \ldots, n^{x_k}$ that is obtained by assigning vertices to components at random. If $\delta$ and $\epsilon_i$ are positive real numbers, then the expected number of queries used by a deterministic algorithm against $P_2 = P(\epsilon_1 + \delta, \ldots, \epsilon_k + \delta)$ is at least the minimum expected number of queries used by a deterministic algorithm against $P(\epsilon_1, \ldots, \epsilon_k)$.*

*Proof.* Let $A$ be an algorithm against $P_2$; we use $A$ to specify an algorithm against $P_1$. Given $G_1$ drawn from $P_1$, for each vertex $v_i \in V_1$ we add a clique $Q_i$ of $n^\delta$ dummy vertices and an edge between $v_i$ and $Q_i$. Finally, we apply a randomly-generated permutation $\sigma$ to the resulting set of vertices to obtain a graph $G_2$. This permutation guarantees that the distribution of the resulting graphs is that of $P_2$.

The new graph $G_2$ has components of sizes $n^{\epsilon_1+\delta}, \ldots, n^{\epsilon_k+\delta}$. We apply $A$ to $G_2$ to decide what query to make at each step, but we use the oracle for $G_1$ to obtain the responses. In other words, when $A$ wants to query $S_2$, we construct:

$$S_1 = \sigma^{-1}(S_2 \cap \sigma(V_1)) \cup \{v_i \in V_1 \colon \exists u \in S_2 \text{ such that } u \in \sigma(Q_i)\}$$

and send the query $S_1$ to the oracle for $G_1$. After receiving $Q(S_1)$ from this oracle, we give $A$ the following set as the answer of $Q(S_2)$:

$$\sigma(Q(S_1)) \cup \bigcup_{v_i \in Q(S_1)} \sigma(Q_i).$$

Once we know the components of $G_2$, we immediately know the components of $G_1$. Each time we generate a graph $G_1$ from $P_1$, we obtain a graph $G_2$ generated from $P_2$, and the number of queries we use to find the components of $G_1$ is the same as the number of queries used to find the components of $A$. Hence the expected cost in using $A$ against $P_2$ is at least the optimal cost against $P_1$ □

Let $\pi$ be a permutation of $\{1, 2, \ldots, k\}$ such that

$$\epsilon_{\pi(1)} < \epsilon_{\pi(2)} < \cdots < \epsilon_{\pi(k)}.$$

We call each interval $[\epsilon_{\pi(i)}, \epsilon_{\pi(i+1)})$ an $\epsilon$-interval. The length of $[\epsilon_{\pi(i)}, \epsilon_{\pi(i+1)})$ is $\epsilon_{\pi(i+1)} - \epsilon_{\pi(i)}$. The sequence of real values $\epsilon_{\pi(1)}, \ldots, \epsilon_{\pi(k)}$ partition $[0, 1)$ into $k - 1$ disjoint $\epsilon$-intervals.

**Lemma 9.** *Let $x$ be any real number in $[0, 1)$. Then*

$$\Pr[x \textit{ is in an } \epsilon\textit{-interval of length } \tfrac{1}{2^i}] = \frac{1}{2^i}$$

*for any $1 \leq i < k - 2$.*

*Proof.* Of the $k - 1$ $\epsilon$-intervals, there is one with length $2^{-i}$ for each $i = 1, 2, \ldots, k-3$, and two with length $2^{-(k-2)}$ that are separated by $\epsilon_k$. This claim is true by force when $k = 3$, and we proceed by induction for larger $k$. When we generate $\epsilon_k$, we move halfway into one of the smallest $\epsilon$-intervals formed by the first $k - 1$ values, and the distance we move splits it in half. □

**Lemma 10.** *Let $T(n, k)$ be the number of queries used by a deterministic algorithm $A$ on input distribution (1). Then $E[T(n, k)] = \Omega(\min\{k, \log\log n\})$.*

*Proof.* Let $S$ be the set of vertices the deterministic algorithm picks to make the first query. Assume without loss of generality that $S \neq V$. Let $x = \log_n \frac{n}{|S|} \in [0, 1)$. The expected number of vertices of component $i$ that are in $S$ will be

$$\frac{n^{\epsilon_i}}{n} \cdot |S| = \frac{n^{\epsilon_i}}{n^x}.$$

It is easy to show that if $\epsilon_i > x$, then component $i$ will be in $Q(S)$ with high probability, and if $\epsilon_i < x$, then component $i$ will be in $V - Q(S)$ with high probability. Since $x \in [0, 1)$, there must exist an $\epsilon$-interval $[\epsilon_{\pi(i)}, \epsilon_{\pi(i+1)})$ that contains $x$. Therefore after the first query, we are left with two disjoint sets of vertices $Q(S)$ and $V - Q(S)$, where $Q(S)$ contains large components $C_{\pi(i+1)}, C_{\pi(i+2)}, \ldots, C_{\pi(k)}$, and $V - Q(S)$ contains small components $C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(i)}$.

Let the length of the $\epsilon$-interval $[\epsilon_{\pi(i)}, \epsilon_{\pi(i+1)})$ be $2^{-j}$, then we must have $\pi(i) = j+2$ or $\pi(i+1) = j+2$. If $\pi(i) = j+2$, then

$$\epsilon_{j+2} - \frac{1}{2^j} < \epsilon_{j+3}, \epsilon_{j+4}, \ldots, \epsilon_k < \epsilon_{j+2}.$$

Therefore $V - Q(S)$ contains $i$ components of sizes

$$n^{\epsilon_{\pi(1)}}, n^{\epsilon_{\pi(2)}}, \ldots, n^{\epsilon_{\pi(i)}}.$$

Among the $i$ components, the following $k-j-1$ components form a small instance of the same input distribution (1):

$$n^{\epsilon_{j+2}}, n^{\epsilon_{j+3}}, \ldots, n^{\epsilon_k}. \tag{2}$$

On the other hand, if $\pi(i+1) = j+2$, then

$$\epsilon_{j+2} < \epsilon_{j+3}, \epsilon_{j+4}, \ldots, \epsilon_k < \epsilon_{j+2} + \frac{1}{2^j}.$$

Therefore $Q(S)$ will contain $k-i$ components of sizes

$$n^{\epsilon_{\pi(i+1)}}, n^{\epsilon_{\pi(i+2)}}, \ldots, n^{\epsilon_{\pi(k)}}$$

and $S$ contains the same $k-i$ components but the sizes are smaller:

$$n^{\epsilon_{\pi(i+1)}-x}, n^{\epsilon_{\pi(i+2)}-x}, \ldots, n^{\epsilon_{\pi(k)}-x}.$$

Among the $k-i$ components, the following $k-j-1$ components form a small instance of the input distribution (1):

$$n^{\epsilon_{j+2}-x}, n^{\epsilon_{j+3}-x}, \ldots, n^{\epsilon_k-x}. \tag{3}$$

According to Lemma 8, finding the components of $Q(S)$ is as hard as finding the components of the graph induced by $S$, whose components have sizes given by (3).

The first query leaves us with one of two sub-problems of the types described above By Lemma 9, the probability is $2^{-j}$ that one of these contains a problem with $k-j-1$ components as described in (2) or (3). These are smaller instances of problem (1). The numbers of vertices in (2) and (3) are approximately $n^{-2^j}$. Therefore we have the following recurrence relation:

$$E[T(n,k)] \geq 1 + \sum_{j=1}^{k} \frac{1}{2^j} E[T(n^{\frac{1}{2^j}}, k-j-1)] \tag{4}$$

Solving (4) by induction, the Lemma is proved. □

¿From Yao [10], Lemma 10 implies a randomized lower bound. Since $\log k$ is always a lower bound, we have

**Theorem 11.** *On $n$-vertex graphs with $k$ components, the expected cost of every randomized algorithm is at least $\Omega(\min\{k, \log k + \log\log n\})$ queries.*

*Acknowledgments* The research of Weiping Shi was supported in part by NSF grant MIP-9309120. Douglas West was supported by NSA/MSP grant MDA904-90-H-4011. The authors wish to thank Neal Brand, Edward Reingold, and Steve Tate for discussions and suggestions. Thanks also go to three anonymous program committee members for pointing out errors in an earlier version of the paper and for improving the presentation.

## References

1. C. C. Chen and F. Hwang. Detecting and locating electrical shorts using group testing. *IEEE Trans. on Circuits and Systems* 36 (8), pp. 1113-1116, Aug. 1989.
2. W.-T. Cheng, J. L. Lewandowski and E. Wu. Optimal diagnostic methods for wiring interconnects. *IEEE Trans. on Computer-Aided Design* 11 (9), pp. 1161–1166, Sept. 1992.
3. A. Fiat, D. P. Foster, H. Karloff, Y. Rabani, Y. Ravid, and S. Vishwanathan. Competitive algorithms for layered graph traversal. *FOCS*, pp. 288–297, 1991.
4. M. Garey, D. Johnson, and H. So. An application of graph coloring to printed circuit testing. *IEEE Trans. Circuits and Systems* 23 (10), pp. 591–599, Oct. 1976.
5. N. Jarwala and C. W. Yau. A new framework for analyzing test generation and diagnosis algorithms for wiring interconnect. *Proc. International Testing Conference*, pp. 63–70, 1989.
6. R. M. Karp. Probabilistic recurrence relations. *J. of ACM* 41 (6), Nov. 1994, pp. 1136–1150.
7. W. H. Kautz. Testing for faults in wiring networks. *IEEE Trans. on Computers,* 23 (4), pp. 358–363, April 1973.
8. L. Kavraki, J.-C. Latombe, R. Motwani and P. Raghavan. Randomized query processing in robot motion planning. *STOC*, 1995.
9. W. Shi and W. K. Fuchs. Optimal interconnect diagnosis of wiring networks. To appear *IEEE Trans. on VLSI Systems*, Sept. 1995.
10. A. Yao, Probabilistic Computations: Towards a Unified Measure of Complexity. *Proc. FOCS*, pp. 222–227, 1977.