SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853-3801

TECHNICAL REPORT NO. 993

December 1991

SCAN-FIRST SEARCH AND SPARSE CERTIFICATES: AN IMPROVED PARALLEL ALGORITHM FOR K-VERTEX CONNECTIVITY

by
Joseph Cheriyan[1], Ming-Yang Kao[2]
and Ramakrishna Thurimella[3]

[1]Research supported in part by the NSF, the AFOSR, and the ONR, through NSF grant DMS-8920550, and by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).

[2]Department of Computer Science, Duke University, Durham, NC 27706. Research supported in part by NSF Grant CCR-9101385.

[3]Department of Mathematics and Computer Science, University of Denver, Denver, CO 80208. This work was done while this author was with UMIACS, University of Maryland at College Park, College Park, MD 20742.

SCAN-FIRST SEARCH AND SPARSE CERTIFICATES: AN IMPROVED PARALLEL ALGORITHM FOR K-VERTEX CONNECTIVITY

JOSEPH CHERIYAN*, MING-YANG KAO† AND RAMAKRISHNA THURIMELLA‡

Abstract. Given a graph $G = (V, E)$, a *certificate* of k-vertex connectivity is an edge subset $E' \subset E$ such that the subgraph (V, E') is k-vertex connected if and only if G is k-vertex connected. Let n and m denote the number of vertices and edges. A certificate is called *sparse* if it contains $O(kn)$ edges.

For undirected graphs, we introduce a graph search called the *scan-first search*, and show that a certificate with at most $k(n-1)$ edges can be computed by executing scan-first search k times in sequence on subgraphs of G. For each of the parallel, distributed and sequential models of computation, the complexity of scan-first search matches the best complexity of any graph search on that model. In particular, the parallel scan-first search runs in $O(\log n)$ time using $C(n, m)$ processors on a CRCW PRAM, where $C(n, m)$ is the number of processors needed to find a spanning tree in each connected component in $O(\log n)$ time, and the parallel certificate algorithm runs in $O(k \log n)$ time using $C(n, m)$ processors. Our parallel certificate algorithm can be employed to test the k-vertex connectivity of an undirected graph in $O(k^2 \log n)$ time using $k \cdot n \cdot C(n, kn)$ processors on a CRCW PRAM. For all combinations of n, m, and $k > 3$, both the running time and the number of processors either improve on or match those of all known deterministic parallel algorithms.

We also obtain an online algorithm for computing an undirected graph certificate with at most $2kn$ edges, and a sequential algorithm for computing a directed graph certificate with at most $2k^2n$ edges.

1. Introduction. Graph connectivity is one of the most fundamental properties in graph theory [4] . Given a positive integer k, an undirected (respectively, directed) graph $G = (V, E)$ with at least $k+1$ vertices is called *k-vertex connected* if the deletion of any $k-1$ vertices leaves the graph connected (respectively, strongly connected). A *certificate* for the k-vertex connectivity of G is a subset E' of E such that the subgraph (V, E') is k-vertex connected if and only if G is k-vertex connected. Let $n = |V|$ and $m = |E|$. Note that $kn/2$ is a trivial lower bound on the number of edges in a certificate for a k-vertex connected graph. For general k, it is not obvious that there is any upper bound strictly less than m on the number of edges in a certificate for k-vertex connectivity; however, nonconstructive results of Mader imply an upper bound of $O(kn)$ for undirected graph certificates [5]. We call a certificate for k-vertex connectivity *sparse* if it has $O(kn)$ edges. For instance, a spanning tree is a sparse certificate for the 1-vertex connectivity of a connected undirected graph.

Sparse certificates have applications in diverse areas of computer science. For example, they can be used for message-efficient fault-tolerant protocols in distributed

* O. R. and I. E., Cornell University, Ithaca, NY 14853. Research supported in part by the NSF, the AFOSR, and the ONR, through NSF grant DMS-8920550, and by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).

† Department of Computer Science, Duke University, Durham, NC 27706. Research supported in part by NSF Grant CCR-9101385.

‡ Department of Mathematics and Computer Science, University of Denver, Denver CO 80208. This work was done while this author was with UMIACS, University of Maryland at College Park, College Park, MD 20742.

computing [20], [21]. Also, they are useful for improving existing graph k-vertex connectivity algorithms. The k-vertex connectivity of a graph can be tested in two stages. Stage 1 computes a sparse certificate of the input graph. Stage 2 applies a given k-vertex connectivity algorithm to the certificate obtained in Stage 1. Because a certificate preserves the k-vertex connectivity of the input graph, Stage 1 ensures the correctness of the test. Stage 2 can *potentially* improve the complexity of the test by reducing the size of the input to the given k-vertex connectivity algorithm.

For sequential computing, Nagamochi and Ibaraki have presented an algorithm for sparse undirected graph certificates that runs in $O(m+n)$ time [26]. For testing the k-vertex connectivity of undirected graphs, this gives sequential running times of $O(k^2n^2)$ for $k \leq \sqrt{n}$ and of $O(k^3n^{1.5})$ for $k > \sqrt{n}$ when their sparse-certificate algorithm is combined with Galil's k-vertex connectivity algorithm [17]. Independently, another sequential algorithm for k-vertex connectivity with the same complexity for $k \leq \sqrt{n}$ was reported in preliminary versions of this paper [9], [8], the stated bound being achieved in the latter [8]. It is now clear that the work of Nagamochi and Ibaraki [26] was the earlier result, although unknown to us at the time of our work.

We present an algorithm for finding sparse certificates for undirected graphs, using a new graph search procedure called the *scan-first search*. This search procedure has surprisingly efficient implementations in the parallel, distributed and sequential models of computation. Consequently, our certificate algorithm also has efficient implementations in all these three models. Below we list the complexity of scan-first search on the three models. Note that the complexity of scan-first search on each model matches the best complexity of any graph search on that model.

- **Parallel Computing.** Scan-first search runs in $O(\log n)$ time using $C(n,m)$ processors on a CRCW PRAM, where $C(n,m)$ is the number of processors used to compute a spanning tree in each connected component in $O(\log n)$ time. For deterministic algorithms, Cole and Vishkin have shown [11] that $C(n,m) = O((m+n)\alpha(n,m)/\log n)$, where $\alpha(n,m)$ is the inverse of Ackerman's function and has an extremely slow growth rate [12]. For randomized algorithms, Gazit has shown [18] that $C(n,m)$ achieves the optimal bound of $\Theta((m+n)/\log n)$.
- **Distributed Computing.** Scan-first search runs in $O(d \cdot \log^3 n)$ time using $O(m + n\log^3 n)$ messages, where d is the diameter of the input graph.
- **Sequential Computing.** Scan-first search runs in $O(m+n)$ time.

The advantage of scan-first search over the two most well-known graph search procedures is easy to see. Depth-first search runs in optimal linear time on the sequential model [27] but has no known parallel implementation that is efficient [1], [2]. Breadth-first search runs efficiently on the sequential and the distributed models [14], [3] but currently the best parallel implementation is no more efficient than matrix multiplication [19].

For undirected graphs, we show that a sparse certificate can be computed by executing scan-first search k times in sequence on subgraphs of G; moreover, the resulting certificate has at most $k(n-1)$ edges, only a factor of two away from the trivial lower bound (see Theorem 2.4). Combining this result with the above implementations of

scan-first search shows that the complexity of our sparse-certificate algorithm on the three models is as follows: $O(k \log n)$ time using $C(n,m)$ processors on the parallel model, $O(k{\cdot}n{\cdot}\log^3 n)$ time using $O(k(m+n\log^3 n))$ messages on the distributed model, and $O(k(m+n))$ time on the sequential model.

Consider the problem of testing an undirected graph for k-vertex connectivity. Using the method sketched above, our certificate algorithm can be used as a preprocessing step, both with Khuller and Schieber's parallel algorithm [24], and with Even's sequential algorithm [13]. This gives a parallel running time of $O(k^2 \log n)$ using $k{\cdot}n{\cdot}C(n,\min\{kn,m\})$ processors, and sequential running times of $O(k^2n^2)$ when $k \leq \sqrt{n}$ and of $O(k^3n^{1.5})$ when $k > \sqrt{n}$.

For general k, there are no previous parallel or distributed algorithms for sparse certificates. The best previous parallel algorithm for (undirected) k-vertex connectivity, due to Khuller and Schieber [24], runs in $O(k^2 \log n)$ time and uses $k{\cdot}n{\cdot}C(n,m)$ processors. For $m > kn$, our algorithm uses fewer processors than their algorithm. For dense graphs and for $k = O(1)$, our algorithm runs in $O(\log n)$ time and has a time-processor product that is within an alpha-factor of the trivial lower bound of $\Omega(n^2)$.

We also obtain the following results. For *undirected* graphs, we give an *online* algorithm that computes a certificate with at most $2kn$ edges. This algorithm is parallelized to run probabilistically in $O(\log^2 n)$ time using $m{\cdot}P(n,m)$ processors, where $P(n,m)$ is the number of processors needed to find a maximum matching in $O(\log^2 n)$ time with high probability. Currently, the best value known for $P(n,m)$ is $O(m{\cdot}n^{3.38})$ [25]. For *directed* graphs, we show that a certificate with at most $2k^2n$ edges can be computed in $O(k{\cdot}m{\cdot}\max\{n,k\sqrt{n}\})$ sequential time.

The above discussion has highlighted the results of this paper. The following sections proceed to detail those results. Section 2 discusses scan-first search and the algorithm for computing undirected graph certificates in the three models. Section 3 presents the online algorithm for undirected graph certificates and its parallelization. Section 4 describes the sequential algorithm for directed graph certificates. Section 5 concludes the paper with open problems.

2. Scan-first search and sparse undirected graph certificates. The main result of this section is that a sparse certificate for undirected k-vertex connectivity can be found by iteratively performing k scan-first searches on subgraphs of the input graph.

Section 2.1 defines scan-first search and discusses how to perform the search efficiently in the parallel, distributed, and sequential models of computation. Section 2.2 states the main certificate theorem based on scan-first search and discusses its algorithmic implications. Section 2.3 proves the main certificate theorem, and Section 2.4 gives a generalization of the theorem.

2.1. Scan-first search. Given a connected undirected graph and a specified vertex, a *scan-first search* in the graph starting from the specified vertex is a systematic way of marking the vertices. The main marking step is called *scan*: to scan a marked vertex means to mark all previously unmarked neighbors of that vertex. At the be-

Subroutine PARALLEL SCAN-FIRST SEARCH
Input: a connected undirected graph $G = (V, E)$ and a vertex r.
Output: a scan-first search spanning tree T of G rooted at r.
begin
Find a spanning tree T' of G rooted at r;
Assign a preorder numbering to the vertices in T';
For each vertex $v \in T'$ with $v \neq r$, let $b(v)$ denote the neighbor of v in G with the smallest preorder number;
Let T be the tree formed by the edges $\{v, b(v)\}$ for all $v \neq r$;
end.

FIG. 1. *Computing a scan-first search spanning tree in parallel.*

ginning of the search, only the specified starting vertex is marked. Then, the search iteratively scans a marked and unscanned vertex until all vertices are scanned.

A scan-first search in a connected undirected graph produces a spanning tree defined as follows. At the beginning of the search, the tree is empty. Then, for each vertex x in the graph, when x is scanned, all the edges between x and its previously unmarked neighbors are added to the tree; the edges between x and its previously marked neighbors are not added to the tree.

For an undirected graph that may or may not be connected, a scan-first search can be performed by applying the above search procedure to each connected component (as well as to each isolated vertex). The search produces a spanning forest with a spanning tree in each connected component.

Notice that scan-first search is more general than sequential breadth-first search [14]. In other words, all sequential breadth-first search trees are scan-first search trees but some scan-first search trees are not breadth-first search trees. For example, let C be an undirected graph consisting of a five-vertex cycle x_1, x_2, x_3, x_4, x_5. Let e_i denote the edge $\{x_i, x_{i+1}\}$. Then, $C - \{e_2\}$, $C - \{e_3\}$, $C - \{e_4\}$ are the scan-first search trees of C rooted at x_1. However, $C - \{e_3\}$ is the only breadth-first search tree rooted at x_1.

THEOREM 2.1. *For an undirected graph with n vertices and m edges, a scan-first search spanning forest can be found in $O(\log n)$ time using $C(n, m)$ processors on a CRCW PRAM, where $C(n, m)$ is the number of processors used to compute a spanning tree in each connected component in $O(\log n)$ time.*

Proof. Let G be the input graph. Without loss of generality, assume that G is connected. Let r be a given starting vertex in G. To prove the theorem, Figure 1 describes an algorithm for finding a scan-first search spanning tree T of G rooted at the vertex r.

T is a scan-first search spanning tree of G rooted at r for the following reasons. T corresponds to a scan-first search in G starting at r with the preorder of T' being the scanning order. T is a spanning tree because every vertex except r has a neighbor with a smaller preorder number in T'.

T can be found in $O(\log n)$ time using $C(n, m)$ processors on a CRCW PRAM as follows. By the definition of $C(n, m)$, T' can be found in $O(\log n)$ time using $C(n, m)$

processors. By standard parallel algorithmic techniques [23], the preorder numbers and the neighbors $b(v)$ in Figure 1 can be computed in $O(\log n)$ time using $O((n+m)/\log n)$ processors. Because $C(n,m) = \Omega((n+m)/\log n)$, the total complexity is $O(\log n)$ time using $C(n,m)$ processors. □

To find a scan-first search tree on the distributed model, we simply use the best distributed breadth-first search algorithm currently known, that of Awerbuch and Peleg [3].

THEOREM 2.2. *For an undirected graph with diameter d, n vertices, and m edges, a scan-first search spanning forest can be found in $O(d \cdot \log^3 n)$ distributed time using $O(m + n\log^3 n)$ messages.*

Proof. The distributed breadth-first search algorithm of Awerbuch and Peleg [3] runs in $O(d \cdot \log^3 n)$ distributed time using $O(m + n\log^3 n)$ messages. We first execute their algorithm, and then modify the resulting breadth-first search forest to give a scan-first search forest within the same complexity bounds. □

THEOREM 2.3. *For an undirected graph with n vertices and m edges, a scan-first search spanning forest can be found in $O(n+m)$ sequential time.*

Proof. Easy. □

2.2. The main certificate theorem and its algorithmic implications. The next theorem shows that sparse certificates for the k-vertex connectivity of undirected graphs can be computed efficiently.

THEOREM 2.4 (THE MAIN CERTIFICATE THEOREM). *Let $G = (V,E)$ be an undirected graph and let n denote the number of vertices. Let k be a positive integer. For $i = 1, 2, \cdots, k$, let E_i be the edge set of a scan-first search forest in the graph $G_{i-1} = (V, E - (E_1 \cup \cdots \cup E_{i-1}))$. Then $E_1 \cup \cdots \cup E_k$ is a certificate for the k-vertex connectivity of G, and this certificate has at most $k(n-1)$ edges.*

Theorem 2.4 has algorithmic consequences for the parallel, distributed and sequential models of computation.

THEOREM 2.5. *For an undirected graph with n vertices and m edges, a sparse certificate for k-vertex connectivity with at most $k(n-1)$ edges can be found in $O(k \log n)$ time using $C(n,m)$ processors on a CRCW PRAM.*

Proof. By Theorems 2.4 and 2.1. □

The next theorem improves on the best previous parallel algorithms for testing the k-vertex connectivity of undirected graphs, namely, the algorithm in Khuller and Schieber [24] and the one in the preliminary version of this paper [10].

THEOREM 2.6. *For an undirected graph with n vertices and m edges, the k-vertex connectivity can be tested in $O(k^2 \log n)$ time using $k \cdot n \cdot C(n, kn)$ processors on a CRCW PRAM.*

Proof. The k-vertex connectivity is tested in two steps. Step 1: Compute a sparse certificate for the k-vertex connectivity of the input graph via Theorem 2.5. Step 2: Apply the k-vertex connectivity algorithm of Khuller and Schieber [24] to the certificate found in the first step. Step 1 runs in $O(k \log n)$ time using $C(n,m)$ processors. Step 2 runs in $O(k^2 \log n)$ time using $k \cdot n \cdot C(n, kn)$ processors. Because $k \cdot n \cdot C(n, kn) \geq C(n,m)$, the total complexity is as stated. □

The main certificate theorem also gives an efficient algorithm on the distributed model of computation. For general k, the first distributed algorithm for undirected sparse certificates was presented in the preliminary version of this paper [10]. For $k = 2$, Itai and Rodeh have previously given a distributed algorithm for undirected sparse certificates [21].

THEOREM 2.7. *For an undirected graph with n vertices and m edges, a sparse certificate for k-vertex connectivity with at most $k(n-1)$ edges can be found in $O(k \cdot n \cdot \log^3 n)$ distributed time using $O(k(m + n \log^3 n))$ messages.*

Proof. Follows from Theorems 2.4 and 2.2. For $i > 1$, notice that the diameter of G_{i-1} may increase to $\Omega(n)$. □

For sequential computation, linear-time algorithms for sparse certificates for the 2-vertex connectivity and the 3-vertex connectivity of undirected graphs have been given by Itai and Rodeh [21] and Cheriyan and Maheshwari [7], respectively. For general k, Nagamochi and Ibaraki have recently given a linear-time algorithm for sparse certificates for undirected graphs [26]. The main certificate theorem gives a slower sequential algorithm for general k.

THEOREM 2.8. *For an undirected graph with n vertices and m edges, a sparse certificate for k-vertex connectivity with at most $k(n-1)$ edges can be found in $O(k(m+n))$ sequential time.*

Proof. By Theorems 2.4 and 2.3. □

For testing the k-vertex connectivity of undirected graphs, sequential running times of $O(k^2n^2)$ for $k \leq \sqrt{n}$ and of $O(k^3n^{1.5})$ for $k > \sqrt{n}$ have been reported in Nagamochi and Ibaraki [26]. The main certificate theorem gives the same sequential complexity, when combined with Even's k-vertex connectivity algorithm [13].

THEOREM 2.9. *The k-vertex connectivity of an n-vertex undirected graph can be tested in $O(k^2n^2)$ sequential time for $k \leq \sqrt{n}$ and in $O(k^3n^{1.5})$ sequential time for $k > \sqrt{n}$.*

Proof. First use Theorem 2.8 to find a sparse certificate. Then run the k-vertex connectivity algorithm of Even [13], [14]. □

COROLLARY 2.10. *For $k = O(1)$, the k-vertex connectivity of an n-vertex undirected graph can be tested in $O(n^2)$ sequential time.*

Proof. Follows from Theorem 2.9. □

2.3. The proof of the main certificate theorem. The main certificate theorem states that a certificate with at most $k(n-1)$ edges can be computed by successively finding the edge set E_1 of a scan-first search spanning forest of $G_0 = G$, the edge set E_2 of a scan-first search spanning forest of $G_1 = (V, E - E_1)$, ..., the edge set E_k of a scan-first search spanning forest of $G_{k-1} = (V, E - (E_1 \cup \cdots \cup E_{k-1}))$, and taking their union $E_1 \cup \cdots \cup E_{k-1}$.

For $i = 1, \cdots, k$, let F_i denote the spanning forest computed by the ith scan-first search (i.e., F_i has edge set E_i), and let H_i denote the subgraph $(V, (E_1 \cup \cdots \cup E_i))$. See Figure 4 for an example illustrating the definitions of G_i, F_i and H_i. Clearly, the theorem holds if H_k is k-vertex connected. To prove the theorem by contradiction, suppose that H_k is not k-vertex connected, and that G is k-vertex connected. Then

there is a subset S of at most $k-1$ vertices such that $H_k - S$ is disconnected, by Menger's theorem. The next lemma shows that at least one tree of the last scan-first search forest F_k must contain vertices of two or more connected components of $H_k - S$.

LEMMA 2.11. *Suppose that H_k is not k-vertex connected, and that G is k-vertex connected. Then the following two statements hold.*

1. *There is a subset $S \subset V$ with $|S| < k$ such that $H_k - S$ is disconnected.*
2. *F_k contains a simple tree path P_k whose two endpoints are in different connected components of $H_k - S$.*

Proof. Focus on the second statement. By the k-vertex connectivity of G, the graph $G - S$ obtained by deleting S from G is connected because S contains $k-1$ or fewer vertices. Because $H_k - S$ is disconnected and $G - S$ is connected, there exists an edge e in G whose endpoints are in two different connected components of $H_k - S$. The edge e is not in H_k, and so is not in $E_1 \cup \cdots \cup E_{k-1}$. Hence the edge e is in $G_{k-1} = (V, E - (E_1 \cup \cdots \cup E_{k-1}))$, and so the two endpoints of e are in the same connected component of G_{k-1}. Because F_k is a scan-first search forest in G_{k-1}, the forest F_k has a spanning tree for the connected component in G_{k-1} that contains e. Therefore, F_k contains a simple tree path P_k between the two endpoints of e. This shows that the second statement holds. □

To finish the proof of Theorem 2.4, the following discussion proceeds to show that the path P_k of Lemma 2.11 cannot exist, yielding the desired contradiction.

A few definitions are in order. Let ω denote the size of S. The proof of Theorem 2.4 makes crucial use of the following indexing scheme of S. Let $s_1, \cdots, s_\omega$ denote the vertices of S such that s_i is the first vertex in $S - \{s_1, \cdots, s_{i-1}\}$ that is scanned by the ith search. Because $\omega < k$ by the first statement of Lemma 2.11, this indexing scheme is well-defined and establishes a one-to-one onto correspondence between the vertices in S and the forests $F_1, \cdots, F_\omega$.

For each s_i in S, the *home component* of s_i is defined as follows. In the forest F_i, let r be the root of the tree that contains s_i. There are three cases. Case (1): If $r \notin S$, then the home component of s_i is the connected component in $H_k - S$ that contains r. Case (2): If $r \in S$ and $r \neq s_i$, then the home component of s_i is the home component of r. Case (3): If $r = s_i$, then s_i has no home component.

Figure 5 illustrates the definition of home components. The next lemma shows that the definition is consistent.

LEMMA 2.12. *For each $s_i \in S$, if s_i satisfies case (1) or case (2) of the above definition, then the home component of s_i is a connected component of $H_k - S$. If s_i satisfies case (3), then s_i has no home component.*

Proof. The lemma is obviously true if s_i satisfies case (1) or case (3) of the definition. For case (2), the lemma is shown by induction on the indices of S.

Induction Hypothesis: For all $j < i$, if s_j satisfies case (2) of the above definition, then the home component of s_j is a connected component of $H_k - S$.

Induction Step: The goal is to show that s_i has a home component. Because s_i satisfies case (2), the root of the tree in F_i that contains s_i is some $s_h \in S$.

CLAIM 1. $h < i$.

Note that $i \neq h$ because $s_h \neq s_i$ by the definition of case (2). To prove this claim by contradiction, assume that $h > i$. Then, $s_h \in S - \{s_1, \cdots, s_{i-1}\}$. Moreover, because s_i is a descendant of s_h in F_i, the ith search scans the vertex s_h before s_i. Thus, the vertex s_i should not have been indexed i. This is a contradiction. Consequently, h must be smaller than i. This finishes the proof of Claim 1.

CLAIM 2. *No tree of F_h has s_h as its root.*

To prove this claim by contradiction, assume that s_h is a root in F_h. Then the edges in G that are adjacent to s_h and are not in $F_1, \cdots, F_{h-1}$ would all be included in F_h. Consequently, s_h would be an isolated vertex in G_h. Because $i > h$ by Claim 1, s_h would be an isolated vertex in F_i. This contradicts the fact that s_i is a descendant of s_h in F_i. Therefore, s_h is not a root in F_h. This finishes the proof of Claim 2.

To show that s_i has a home component, by the definition of case (2), it suffices to prove that s_h has a home component. If case (1) holds for s_h, then by definition, s_h has a home component. If case (2) holds for s_h, then by Claim 1 and the induction hypothesis, s_h has a home component. Case (3) cannot hold for s_h by Claim 2. This finishes the proof of the lemma. □

To describe the key properties of scan-first search, more definitions are needed. For each vertex $s \in S$ if s has a home component, let $hcc(s)$ denote the home component of s; if s has no home component, let $hcc(s)$ denote s itself. Moreover, for each $v \in V - S$, let $hcc(v)$ denote the connected component in $H_k - S$ that contains v.

Given a forest F_i, a *jump* of F_i is a simple tree path $Q = v_1, \cdots, v_q$ of F_i with $hcc(v_1) \neq hcc(v_q)$.

For each vertex $s \in S$, the edges incident with s in G are classified into three types as follows. The *back* edges of s are those between s and its home component. The *side* edges of s are those between s and $S - \{s\}$. The rest of the edges incident with s are the *forward* edges. Note that if s does not have a home component, then it has no back edges. Refer to Figure 5 for an illustration of these definitions.

The next lemma shows a key property of the first scan-first search, and the following lemma, which is the critical one for the proof of the main certificate theorem, generalizes the key property to the first i scan-first searches for all $i \in \{1, 2, \cdots, \omega\}$.

LEMMA 2.13. *The following two statements are true.*

1. *Every jump of F_1 contains at least one vertex of S.*
2. *The scan-first search forest F_1 contains all the forward edges of s_1.*

Proof. To prove the first statement by contradiction, assume that F_1 has a jump $Q = v_1, \cdots, v_q$ which contains no vertices from S. Then, $v_1, v_q \in V - S$. By the definition of hcc, $hcc(v_1)$ and $hcc(v_q)$ are connected components in $H_k - S$. By the definition of a jump, $hcc(v_1) \neq hcc(v_q)$. In sum, Q is a path in H_k that connects two different connected components of $H_k - S$. Therefore, Q must contain a vertex from S, contradicting the assumption of the proof. This finishes the proof of the first statement.

To prove the second statement, note that F_1 is a tree because $G_0 = G$ is connected. Let r be the root of F_1. Then there are two cases: either $r \in S$ or $r \notin S$.

Case (1): $r \in S$. By the definition of the indexing scheme of S, $r = s_1$. When the first search scans s_1, none of the neighbors of s_1 in G have been marked. Consequently,

F_1 includes all edges incident with s_1 in G, and hence the lemma holds for this case.

Case (2): $r \notin S$. Refer to Figure 6(ii) for an illustration of this case. By definition, s_1 has a home component and $hcc(r) = hcc(s_1)$. To prove this case by contradiction, assume that there is a forward edge $e = \{s_1, x\} \notin F_1$. This can happen only if the first search marks x before it scans s_1. Focus on the tree path, say, Q of F_1 from r to x. Because e is a forward edge, $hcc(s_1) \neq hcc(x)$. Therefore, $hcc(r) \neq hcc(x)$ and the path Q is a jump of F_1. By the first statement of this lemma, Q contains a vertex $s \in S$. Because s is an ancestor of x in F_1, the first search scans s before or when it marks x. Hence the first search scans s before it scans s_1. However, this contradicts the definition of the indexing scheme of S, because s_1 should have received a higher index. This finishes the proof of case (2) of the second statement, and completes the proof of the lemma. □

LEMMA 2.14. *For each $i \in \{1, 2, \cdots, \omega\}$, the following statements are true.*

1. *Every jump of F_i contains at least one vertex in $S - \{s_1, \cdots, s_{i-1}\}$.*
2. *$F_1, \cdots, F_i$ contain the following edges of G:*
   (a) *all forward edges of s_i, and*
   (b) *all side edges $\{s_i, s_j\}$ with $i > j$ and $hcc(s_i) \neq hcc(s_j)$.*

Proof. This lemma is proved by induction on i as follows. The base step follows from Lemma 2.13; note that Statement 2(b) holds vacuously for $i = 1$. The induction hypothesis is that the lemma holds for $i = t$. The induction step is to show that the lemma holds for $i = t + 1$. This is proved by the following three claims.

CLAIM 3. *Statement 1 holds for $i = t + 1$.*

To prove the above claim by contradiction, assume that F_{t+1} has a jump $Q = v_1, \cdots, v_q$ which contains no vertices from $S - \{s_1, \cdots, s_t\}$. Refer to Figure 6(i) for an illustration. Let W be the set of vertices that are in both Q and S. Let U_0 be the set of edges in Q that each have two endpoints in $H_k - S$. Let U_1 be the set of edges in Q that each have one endpoint in W and the other endpoint in $H_k - S$. Let U_2 be the set of edges in Q that each have two endpoints in W. Observe that for all edges $\{x, y\} \in U_0$, $hcc(x) = hcc(y)$. Next, from the assumption of the proof, $W \subseteq \{s_1, \cdots, s_t\}$. Therefore, from statement 2(a) of the induction hypothesis, the edges in U_1 cannot be forward edges and hence for all edges $\{x, y\} \in U_1$, $hcc(x) = hcc(y)$. Furthermore, because the edges in U_2 are side edges, from statement 2(b) of the induction hypothesis, for all edges $\{x, y\} \in U_2$, $hcc(x) = hcc(y)$. Hence, for all vertices $x, y \in Q$, $hcc(x) = hcc(y)$. In particular, $hcc(v_1) = hcc(v_q)$, contradicting the assumption that Q is a jump. This finishes the proof of Claim 3.

CLAIM 4. *Statement 2(a) holds for $i = t + 1$.*

To prove the above claim, let T be the tree of F_{t+1} that contains s_{t+1} and let r be the root of T. Then there are two cases: (1) $r \neq s_{t+1}$ or (2) $r = s_{t+1}$.

Case (1): $r \neq s_{t+1}$. Refer to Figure 6(ii) for an illustration of this case. By definition, s_{t+1} has a home component and $hcc(r) = hcc(s_{t+1})$. To prove this case by contradiction, assume that there is a forward edge $e = \{s_{t+1}, x\} \notin F_{t+1}$. This can happen only if the $(t + 1)$-th search marks x before it scans s_{t+1}. From the existence of e, the tree T contains x. So T contains a tree path Q from r to x. Because e is

a forward edge, $hcc(s_{t+1}) \neq hcc(x)$. Therefore, $hcc(r) \neq hcc(x)$ and Q is a jump of F_{t+1}. By Claim 3, Q contains a vertex $s \in S - \{s_1, \cdots, s_t\}$. Notice that $x \neq s$ because $x \in H_k - S$. Therefore, s is an ancestor of x in T. Consequently, the $(t+1)$-th search scans s before or when it marks x. Thus, the $(t+1)$-th search scans s before it scans s_{t+1}. Therefore, s_{t+1} should not have been indexed by $t+1$, contradicting the indexing scheme of S. This finishes the proof of case (1) of Claim 4.

Case (2): $r = s_{t+1}$. Notice that when the $(t+1)$-th search scans s_{t+1}, none of the neighbors of s_{t+1} in G_t have been marked. So F_{t+1} includes all edges incident with s_{t+1} in G_t. Consequently, $F_1, \cdots, F_{t+1}$ include all edges incident with s_{t+1} in G and statement 2(a) holds for case (2). This finishes the proof of case (2) of Claim 4 and the proof of Claim 4.

CLAIM 5. *Statement 2(b) holds for $i = t+1$.*

To prove the above claim, let T be the tree of F_{t+1} that contains s_{t+1} and let r be the root of T. Then there are two cases: (1) $r \neq s_{t+1}$ or (2) $r = s_{t+1}$.

Cases (1): $r \neq s_{t+1}$. Refer to Figure 6(iii) for an illustration of this case. By definition, s_{t+1} has a home component and $hcc(r) = hcc(s_{t+1})$. To prove this case by contradiction, assume that there is a side edge $e = \{s_{t+1}, s_j\} \notin F_{t+1}$ such that $t+1 > j$ and $hcc(s_{t+1}) \neq hcc(s_j)$. This can happen only if the $(t+1)$-th search marks s_j before it scans s_{t+1}. From the existence of e, the tree T contains s_j. So T contains a tree path Q from r to s_j. Notice that $hcc(r) = hcc(s_{t+1}) \neq hcc(s_j)$. Therefore, Q is a jump of F_{t+1}. Next, let W be the set of vertices that are in both S and Q. Notice that $s_j \in W$. Also, s_j is a proper descendant of all vertices in $W - \{s_j\}$, and the $(t+1)$-th search scans $W - \{s_j\}$ before it marks s_j. In sum, the $(t+1)$-th search scans $W - \{s_j\}$ before it scans s_{t+1}. Then, by the indexing scheme of S, $W - \{s_j\} \subseteq \{s_1, \cdots, s_t\}$. By the assumption of the proof, $s_j \in \{s_1, \cdots, s_t\}$. Hence, $W \subseteq \{s_1, \cdots, s_t\}$; by Claim 3, this contradicts the assumption that Q is a jump of F_{t+1}. This finishes the proof of case (1) of Claim 5.

Case (2): $r = s_{t+1}$. This case is exactly the same as case (2) of Claim 4. This finishes the proof of case (2) of Claim 5, the proof of Claim 5, and the proof of the induction step. □

To complete the proof of the main certificate theorem, recall our assumption that H_k is not k-vertex connected, while G is k-vertex connected. Then Lemma 2.11 shows that F_k must contain a path P_k whose endpoints are in two different connected components of $H_k - S$, where S is a subset of V with $|S| < k$ whose deletion disconnects H_k. The next lemma shows that the path P_k cannot exist, using the same argument as in the proof of Claim 3 in Lemma 2.14, and thus completes the proof of Theorem 2.4.

LEMMA 2.15. *The path P_k of the second statement of Lemma 2.11 cannot exist.*

Proof. To prove the lemma by contradiction, assume that P_k exists. Let $P_k = v_1, \cdots, v_q$. Refer to Figure 6(i) for an illustration. Because the two endpoints of P_k are in two different connected components of $H_k - S$, $hcc(v_1) \neq hcc(v_q)$ and the path P_k is a jump of F_k. Let W be the set of vertices that are in both P_k and S. Let U_0 be the set of edges in P_k that each have two endpoints in $H_k - S$. Let U_1 be the set of edges in P_k that each have one endpoint in W and the other endpoint in $H_k - S$. Let

U_2 be the set of edges in P_k that each have two endpoints in W. Observe that for all edges $\{x, y\} \in U_0$, $hcc(x) = hcc(y)$. Next, because $W \subseteq S = \{s_1, \cdots, s_\omega\}$, by statement 2(a) of Lemma 2.14, the edges in U_1 cannot be forward edges and hence for all edges $\{x, y\} \in U_1$, $hcc(x) = hcc(y)$. Furthermore, because the edges in U_2 are side edges, from statement 2(b) of Lemma 2.14, for all edges $\{x, y\} \in U_2$, $hcc(x) = hcc(y)$. In sum, for all vertices $x, y \in P_k$, $hcc(x) = hcc(y)$. In particular, $hcc(v_1) = hcc(v_q)$, contradicting the assumption that P_k is a jump. This finishes the proof of lemma 2.15. □

2.4. A generalization of the main certificate theorem. This section gives a generalization of the main certificate theorem, and discusses its applications.

For two distinct vertices x and y in G, the *local connectivity* of x and y, denoted $\kappa(x, y)$, is the maximum number of internally vertex-disjoint paths between x and y in G. A *certificate of local connectivity* k for G is a subset E' of E such that for every two distinct vertices x and y, $\kappa'(x, y) \geq \min\{k, \kappa(x, y)\}$ where $\kappa'(x, y)$ denotes $\kappa(x, y)$ for the subgraph (V, E'). A certificate of local connectivity k is said to be sparse if it has $O(kn)$ edges.

THEOREM 2.16 (THE GENERALIZED CERTIFICATE THEOREM). *Let $G = (V, E)$ be an undirected graph and let n denote the number of vertices. Let k be a positive integer. For $i = 1, 2, \cdots, k$, let E_i be the edge set of a scan-first search forest in the graph $G_{i-1} = (V, E - (E_1 \cup \cdots \cup E_{i-1}))$. Then $E_1 \cup \cdots \cup E_k$ is a certificate of local connectivity k for G, and this certificate has at most $k(n-1)$ edges.*

Proof. The proof is essentially the same as that of the main certificate theorem. Let H_k be the subgraph $(V, (E_1 \cup \cdots \cup E_k))$, and let $\kappa_k(x, y)$ denote $\kappa(x, y)$ for H_k. To prove the theorem by contradiction, assume that $\kappa_k(u, w) < \min\{k, \kappa(u, w)\}$ for some two vertices $u, w \in V$ with $u \neq w$.

Although Lemma 2.11 does not apply now because it supposes that G is k-vertex connected, we first show that the two statements in Lemma 2.11 hold under the assumption that $\kappa_k(u, w) < \min\{k, \kappa(u, w)\}$ for some two vertices $u, w \in V$ with $u \neq w$. By Menger's theorem, there exist two vertices $u', w' \in V$ and a set $S \subseteq V - \{u', w'\}$ such that (1) $|S| = \kappa_k(u, w)$, (2) u' and w' are in different connected components of $H_k - S$, and (3) u' and w' are in the same connected component of $G - S$. By properties (1) and (2), there is a subset $S \subset V$ with $|S| < k$ such that $H_k - S$ is disconnected. By property (3), there exists an edge e in $G - S$ whose endpoints are in two different connected components of $H_k - S$. Clearly, the edge e is not in H_k, and hence e is in $G_{k-1} = (V, E - (E_1 \cup \cdots \cup E_{k-1}))$. So the two endpoints of e are in the same connected component of G_{k-1}. Because F_k is a scan-first search forest in G_{k-1}, the forest F_k has a spanning tree for the connected component in G_{k-1} that contains e. Therefore, F_k contains a simple tree path P_k whose two endpoints are in different connected components of $H_k - S$. This finishes the proof of the two statements in Lemma 2.11.

With the two statements in Lemma 2.11 proven, we can complete the proof of this theorem by using Lemmas 2.13, 2.14, and 2.15 to show that the above path P_k cannot exist. □

The next corollary is useful for computing the k-separators of a graph. For a positive integer k, a *k-separator* of G is a set S of k vertices such that $G - S$ has more

connected components than G.

COROLLARY 2.17. *For a positive integer $k < n$ and for all $i \in \{1, \cdots, k-1\}$, G and $H_k = (V, (E_1 \cup \cdots \cup E_k))$ have the same i-separators.*

Proof. Straightforward by Theorem 2.16. □

Some of the recent algorithmic research on k-connected graphs has focussed on highly efficient parallel algorithms for finding k-vertex connected components and k-separators for small k, namely, $k = 2$, 3 and 4. Theorem 2.16 and Corollary 2.17, when combined with Theorem 2.5, yield immediate improvements to several of these algorithms.

An undirected graph is said to be *bridge-connected* if for each edge the deletion of that edge leaves the graph connected. For a bridge-connected graph, Fussell and Thurimella [16] have given an algorithm for finding an open ear decomposition for each biconnected component on an $O(\sqrt{n}/\log n \times \sqrt{n}/\log n)$ mesh of trees architecture in $O(\log^3 n)$ time. We improve the running time to $O(\log^2 n)$ by first finding a sparse certificate of local connectivity 2 for the input graph, and then running their algorithm on the subgraph induced by the certificate. The main bottleneck of the original algorithm [16] is to compute, for a given spanning tree T of the input graph, the nearest common ancestor in T of all nontree edges. Note that by executing their algorithm on a sparse certificate, the worst-case number of nontree edges decreases from $\Omega(n^2)$ to $O(n)$.

THEOREM 2.18. *Let $G = (V, E)$ be a bridge-connected graph and let n denote the number of vertices. An open ear decomposition for each biconnected component of G can be found on an $O(\sqrt{n}/\log n \times \sqrt{n}/\log n)$ mesh of trees architecture in $O(\log^2 n)$ time.*

Fussell, Ramachandran and Thurimella [15] have given an algorithm for computing the triconnected components of an undirected graph in $O(\log n)$ time with a time-processor product of $O((m+n)\log\log n)$. We obtain the following improvement by first finding a sparse certificate of local connectivity 3 for the input graph, and then running their algorithm on the subgraph induced by the certificate.

THEOREM 2.19. *Let $G = (V, E)$ be an undirected biconnected graph, and let n and m denote the number of vertices and edges. The 3-vertex connected components of G can be found on an ARBITRARY-CRCW PRAM in $O(\log n)$ time with a time-processor product of $O(m \cdot \alpha(n, m) + n \cdot \log\log n)$.*

For an undirected triconnected graph, Kanevsky and Ramachandran [22] have given an algorithm for finding a compact representation of all the 3-separators. Their algorithm runs in $O(\log^2 n)$ time with a time-processor product of $O(n^2 \log^2 n)$. We obtain the following improvement in two steps. First, find a sparse certificate E' of local connectivity 4 for the input graph $G = (V, E)$. Then, for each vertex v, use the algorithm of Fussell, Ramachandran and Thurimella [15] to find all the 2-separators of $(V, E') - \{v\}$.

THEOREM 2.20. *Let $G = (V, E)$ be an undirected triconnected graph and let n denote the number of vertices. An $O(n^2)$ representation for all the 3-separators of G can be found on an ARBITRARY-CRCW PRAM in $O(\log n)$ time with a time-processor product of $O(n^2 \cdot \log\log n)$.*

The algorithm of Kanevsky and Ramachandran [22] also tests the input graph for

Subroutine ONLINE CERTIFICATE
Input: the edges of an undirected graph $G = (V, E)$ given one at a time in an arbitrary order $e_1, \cdots, e_m$.
Output: a sparse certificate $F \subseteq E$ for the k-vertex connectivity of G.
begin
  $F := \emptyset$;
  for $i = 1$ to m **do**
    begin
      Let v and w denote the endpoints of e_i;
      Let k_i denote the maximum number of vertex-disjoint paths between v and w in (V, F);
      if $k_i < k$ **then** $F := F \cup \{e_i\}$ **else** F remains unchanged;
    end
end.

FIG. 2. *Computing a sparse certificate online.*

4-vertex connectivity in $O(\log^2 n)$ time using $O(n^2)$ processors. Note that for testing 4-vertex connectivity, our Theorem 2.6 gives a running time of $O(\log n)$ using $4 \cdot n \cdot C(n, 4n) = O(n^2 \alpha(n, 4n)/\log n)$ processors.

3. An online algorithm for sparse undirected graph certificates and its parallelization. We first present a sequential online algorithm for finding sparse certificates for the k-vertex connectivity of undirected graphs. Then we parallelize the algorithm to obtain a randomized NC algorithm with a complexity independent of k.

Let $G = (V, E)$ denote the input undirected graph to our online certificate algorithm. Let n and m denote the numbers of vertices and edges of G. G is given one edge at a time; the input order $e_1, \cdots, e_m$ of the edges is arbitrary. Upon receiving an edge e_i, the online certificate algorithm must decide whether to include e_i in the final certificate $F \subseteq E$. Once an edge is included, it cannot be deleted from F at a later step; similarly, if an edge is not included, it cannot be added to F later. Initially, F is empty. Our online algorithm incrementally updates F by examining each input edge $\{v, w\}$ and including $\{v, w\}$ in F if and only if the subgraph induced by the current F has at most $k - 1$ vertex-disjoint paths between v and w.

The detailed description of our online certificate algorithm is given in Figure 2. For an example, refer to Figure 7. The next lemma shows that the F output by the online certificate algorithm is indeed a certificate of k-vertex connectivity.

LEMMA 3.1. *If $G = (V, E)$ is k-vertex connected, then the final subgraph (V, F) is also k-vertex connected.*

Proof. To prove the lemma by contradiction, suppose that G is k-vertex connected but (V, F) is not k-vertex connected. Then, there is a set S of less than k vertices such that $(V, F) - S$ is disconnected. Let I be a connected component of $(V, F) - S$. Since $G - S$ is connected, G has an edge $\{v, w\}$ with $v \in I$ and $w \in V - (I \cup S)$. Because (V, F) has at most $|S|$ vertex-disjoint paths between v and w and because $|S| < k$, the

online certificate algorithm would have added the edge $\{v, w\}$ to F when it examined $\{v, w\}$. This contradicts the assumption of the proof. Thus the lemma is true. □

Next we prove that the final F has at most $2kn$ edges by combining the following lemma with a theorem due to Mader.

LEMMA 3.2. *The final subgraph (V, F) does not contain any subgraph that is $(k+1)$-vertex connected.*

Proof. To prove the lemma by contradiction, suppose that (V, F) contains a $(k+1)$-vertex connected subgraph J. Let $e_i = \{v, w\}$ be the edge with the largest index among all the edges in J. In other words, e_i is the last edge added to J by the online certificate algorithm. Then $J - \{e_i\}$ has k vertex-disjoint paths between v and w because $J - \{e_i\}$ is k-vertex connected. Therefore, when the algorithm examined e_i, it would not have added e_i to F. This contradicts the assumption of the proof and finishes the proof the lemma. □

THEOREM 3.3 (MADER [5]). *For an integer $k \geq 1$, if an undirected graph has at least $2k-1$ vertices and at least $(2k-1)(n-k)+1$ edges, where n denotes the number of vertices, then it contains a $(k+1)$-vertex connected subgraph.*

LEMMA 3.4. *The final F has at most $2kn$ edges.*

Proof. If $n \geq 2k-1$, then the lemma follows from Lemma 3.2 and Theorem 3.3. If $n < 2k-1$, then F contains at most $n(n-1)/2$ edges, which is less than $2kn$. □

The next theorem summarizes the discussion of the online certificate algorithm.

THEOREM 3.5. *Let G be an n-vertex undirected graph. Assume that the edges of G are given one at a time. Then a certificate for the k-vertex connectivity of G with at most $2kn$ edges can be computed on line in $O(k^2 n)$ time per input edge.*

Proof. The correctness of the online certificate algorithm follows from Lemmas 3.1 and 3.4. As for the running time, note that for each edge e_i, the algorithm attempts to find k vertex-disjoint paths between the endpoints of e_i in (V, F). This takes time proportional to k times the size of (V, F) [14]. Therefore, the running time for examining one edge is $O(k^2 n)$. □

A fast parallel version of the online certificate algorithm can be obtained by a parallel greedy method as follows. Let $E_0 = \emptyset$. For $i = 1, \cdots, m$, let E_i denote the edge set $\{e_1, e_2, \cdots, e_i\}$. For $i = 1, \cdots, m$, test in parallel whether the graph (V, E_{i-1}) has at least k vertex-disjoint paths between the endpoints of e_i. The certificate F contains exactly those edges e_i that fail the test.

Notice that the main difference in the computations executed by the above parallel algorithm and the online one is that the maximum number of vertex-disjoint paths between the endpoints of e_i is found in the subgraph (V, E_{i-1}) by the parallel algorithm and in (V, F) by the online algorithm. The next lemma shows that this difference does not affect F.

LEMMA 3.6. *The online certificate algorithm and its parallel version compute the same final F.*

Proof. For $t = 1, \cdots, m$, let F_t denote the F found by the online algorithm after it processes e_t, and let F_0 denote the empty set. Also, let C_p and C_q denote the final F computed by the parallel algorithm and the online algorithm, respectively. The goal

Subroutine FIND-H
Input: a digraph $G = (V, E)$ and a vertex $z \in V$ with $\kappa_G(z, v) \geq k$ for all $v \in V - \{z\}$.
Output: a minimum-size subgraph $H = (V, E')$ with $\kappa_H(z, v) = k$ for all $v \in V - \{z\}$.
begin
Let E' be obtained from E by removing all incoming edges of z;
Let $v_1, v_2, \cdots, v_{n-1}$ be the vertices in $V - \{z\}$;
for $i = 1$ to $n - 1$ **do**
begin
Find k internally vertex-disjoint directed paths from z to v_i in the subgraph (V, E');
Delete from E' all incoming edges of v_i except the k incoming edges in the k paths just found above;
end
end.

FIG. 3. *Computing a subgraph H of Lemma 4.3.*

is show that $C_q = C_p$. Observe that $C_p \subseteq C_q$ because $F_{t-1} \subseteq E_{t-1}$ for all t. Thus, to prove the lemma by contradiction, it suffices to assume that there exists an edge $e_i = \{v, w\} \in C_q - C_p$. Then, by definition, there are at least k vertex-disjoint paths between v and w in (V, E_{i-1}) and there are less than k vertex-disjoint paths between v and w in (V, F_{i-1}). Therefore, there exists a set $S \subseteq V - \{v, w\}$ with $|S| < k$ such that (1) v and w are disconnected in $(V, F_{i-1}) - S$, and (2) there exists an edge $e_j \in E_{i-1} - F_{i-1}$ whose endpoints are in two different connected components of $(V, F_{i-1}) - S$. Notice that $j \leq i-1$ because $e_j \in E_{i-1}$. Moreover, $e_j \notin F_j$ because $e_j \notin F_{i-1}$ and $F_j \subseteq F_{i-1}$. On the other hand, observe that (V, F_{i-1}) contains less than k vertex-disjoint paths between the endpoints of e_j. Then, because $F_{j-1} \subseteq F_{i-1}$, (V, F_{j-1}) also contains less than k vertex-disjoint paths between the endpoints of e_j. But then the online algorithm would have included e_j in F_j. This contradicts the earlier conclusion that $e_j \notin F_j$. Therefore, the assumption of the proof is incorrect and the lemma is correct. □

The next theorem summarizes the discussion of the parallelization of the online certificate algorithm. Let $P(n, m)$ denote the number of processors needed to find a maximum matching in $O(\log^2 n)$ time with high probability. Currently, the best value known for $P(n, m)$ is $O(m \cdot n^{3.38})$ [25].

THEOREM 3.7. *Given an undirected graph with n vertices and m edges, a certificate for k-vertex connectivity with at most $2kn$ edges can be found by a randomized algorithm in $O(\log^2 n)$ time using $m \cdot P(n, m)$ processors on a CRCW PRAM.*

Proof. The correctness of the above parallel algorithm follows from Lemmas 3.6, 3.1, and 3.4. As for the complexity, to test in parallel for the existence of k vertex-disjoint paths between two vertices, we use the well-known method of transforming this problem to the maximum matching problem [6]. □

4. A sequential algorithm for small certificates for directed graphs. The main result of this section is stated in the following theorem.

THEOREM 4.1. *Let $G = (V, E)$ be a directed graph with n vertices and m edges. A certificate for the k-vertex connectivity of G with at most $2k^2n$ edges can be found in $O(k{\cdot}m{\cdot}\max\{n, k\sqrt{n}\})$ sequential time.*

Notice that the running time of our directed graph certificate algorithm is the same as the best time complexity known for testing directed k-vertex connectivity [17]. In contrast with certificates for undirected graphs, relatively little is known about certificates for directed graphs. Ignoring algorithmic issues, to the best of our knowledge, no $O(kn)$ upper bound is known for the minimum size of a certificate for directed k-vertex connectivity. In fact, our algorithm gives the smallest bound known for the minimum size of a certificate. Also, for $k > 1$, our directed graph certificate algorithm is the best algorithm known for finding a certificate of size at most $n{\cdot}k^{O(1)}$.

Our proof of Theorem 4.1 is based on the next lemma. It allows certain edges of a vertex to be deleted without affecting the connectivity elsewhere. For a directed graph G and for every two distinct vertices $x, y \in G$, let $\kappa_G(x, y)$ denote the maximum number of internally vertex-disjoint directed paths from x to y in G.

LEMMA 4.2. *Let $G = (V, E)$ be a directed graph. Let z, u, v be vertices in G. Let e be an incoming edge of u in G. Let $H = (V, E - \{e\})$. If $\kappa_H(z, u) \geq k$ and $\kappa_G(z, v) \geq k$, then $\kappa_H(z, v) \geq k$.*

Proof. To prove the lemma by contradiction, assume that $\kappa_H(z, v) < k$. Then, to derive a contradiction by Menger's theorem, it suffices to show that for every vertex subset $S \subseteq V - \{z, v\}$ with $|S| < k$, there exists a directed path in $H - S$ from z to v.

Because $\kappa_G(z, v) \geq k$, G contains k internally vertex-disjoint directed paths $P_1, \cdots, P_k$ from z to v. Because $\kappa_H(z, v) < k$, the edge e must be in one of the P_i's. Without loss of generality, assume that e is in P_k. Let L be the subpath of P_k from u to v. Note that L is a directed path in H. Also, note that because the P_i's are internally vertex-disjoint, $P_1, \cdots, P_{k-1}$ cannot contain e and therefore remain directed paths in H.

There are two cases based on whether L and S intersect.

Case (1): $L \cap S \neq \emptyset$. Note that $|S - L| \leq k - 2$. Because $P_1, \cdots, P_{k-1}$ are internally vertex-disjoint in H, at least one of those P_i's does not contain any vertices from S, and remains a directed path from z to v in $H - S$. This is a desired contradiction and thus completes the proof of case (1).

Case (2): $L \cap S = \emptyset$. Note that L is directed path in $H - S$ from u to v. Because $\kappa_H(z, u) \geq k$, H contains k internally vertex-disjoint directed paths $Q_1, \cdots, Q_k$ from z to u. Because $Q_1, \cdots, Q_k$ are internally vertex-disjoint and $|S| \leq k - 1$, at least some Q_j does not contain any vertices from S. Then Q_j remains a directed path from z to u in $H - S$. Therefore, the directed path formed by Q_j and L is a directed path from z to v in $H - S$. This is a desired contradiction and thus finishes the proof of case (2). □

The next lemma uses Lemma 4.2 to compute an important subset of a desired certificate for a directed graph. In the lemma, let $G = (V, E)$ be a directed graph with n vertices and m edges. Let z be a vertex in G.

LEMMA 4.3. *Assume that $\kappa_G(z, v) \geq k$ for all $v \in V - \{z\}$. Then there exists a subgraph $H = (V, E')$ computable in $O(k{\cdot}m{\cdot}n)$ time with the following properties:*

1. *for all $v \in V - \{z\}$, $\kappa_H(z, v) = k$,*

2. *the indegree of z in H is zero, and*

3. *for all $v \in V - \{z\}$, the indegree of v in H is exactly k.*

Note that because of the indegree constraints, H is a minimum-size subgraph with $\kappa_H(z, v) = k$ for all $v \in V - \{z\}$.

Proof. H can be computed by the algorithm FIND-H given in Figure 3. Note that deleting the incoming edges of z does not affect $\kappa(z, v)$. Then, the correctness of FIND-H follows directly from repeated applications of Lemma 4.2. As for the time complexity, each iteration of the do loop takes $O(k{\cdot}m)$ time [14]. So the total runing time is $O(k{\cdot}m{\cdot}n)$ □

Our directed graph certificate combines the subroutine FIND-H and the classic k-vertex connectivity algorithm of Even [14]. The certificate algorithm computes a certificate F for the input graph G in five steps as follows:

1. Test the k-vertex connectivity of G. If G is not k-vertex connected, then let $F = \emptyset$ and stop; otherwise, continue the computation.
2. Pick k arbitrary vertices $x_1, \cdots, x_k \in V$. For all x_i and x_j with $i < j$, compute k internally vertex-disjoint directed paths in G from x_i to x_j and k internally vertex-disjoint directed paths from x_j to x_i. Let E_0 denote the set of the edges in all those paths.
3. Let G^+ be the graph constructed by adding to G a new vertex z and $2k$ new edges $(z, x_1), \cdots, (z, x_k)$ and $(x_1, z), \cdots, (x_k, z)$.
4. Use FIND-H to find a subgraph $H_1 = (V \cup \{z\}, E_1)$ for G^+ and z. Symmetrically, find a subgraph $H_2 = (V \cup \{z\}, E_2)$ for G^+ and z with respect to the *reverse* edge and path directions.
5. Let the final certificate F be $E_0 \cup E_1 \cup E_2$ without the edges incident with z.

The next two lemmas show the correctness of the above certificate algorithm.

LEMMA 4.4. *If G is k-vertex connected, then the subgraph (V, F) computed above is k-vertex connected.*

Proof. Let $C = (V, F)$. Let $C^+ = (V \cup \{z\}, E_0 \cup E_1 \cup E_2)$. Note that C^+ is the graph constructed by adding to C the vertex z and the $2k$ edges $(z, x_1), \cdots, (z, x_k)$ and $(x_1, z), \cdots, (x_k, z)$. Next, because $E_1 \cup E_2$ is included in C^+, by Lemma 4.3, $\kappa_{C^+}(z, v) \geq k$ and $\kappa_{C^+}(v, z) \geq k$ for all $v \in V$. Because E_0 is included in C, for all x_i and x_j with $i < j$, $\kappa_C(x_i, x_j) \geq k$ and $\kappa_C(x_j, x_i) \geq k$. Therefore, by the classic argument of Even for his k-vertex connectivity algorithm, (V, F) is indeed k-vertex connected. □

LEMMA 4.5. $|F| \leq 2k^2n$.

Proof. If G is not k-vertex connected, then $|F| = 0$. Otherwise, the upper bound of $|F|$ follows from the facts that $|E_1| = |E_2| = kn$ and $|E_0| \leq k(k-1)(n-2+k)$. □

The next lemma finishes the proof of Theorem 4.1.

LEMMA 4.6. *The above certificate algorithm runs in $O(k{\cdot}m{\cdot}\max\{n, k\sqrt{n}\})$ time.*

Proof. Testing k-vertex connectivity takes $O(k{\cdot}m{\cdot}\max\{n, k\sqrt{n}\})$ time [14]. Computing E_0 takes $O(k^2{\cdot}m{\cdot}\min\{k, \sqrt{n}\})$ time. By Lemma 4.3, computing E_1 and E_2 takes $O(k{\cdot}m{\cdot}n)$ time. These three steps dominate the time complexity. Therefore, the total running time is $O(k{\cdot}m{\cdot}\max\{n, k\sqrt{n}\})$. □

5. Conclusions and open problems. Designing graph search procedures that are efficient in several major models of computation is an important issue in algorithmic graph theory. We have shown that scan-first search can be performed extremely efficiently in the parallel, the distributed, and the sequential models. Based on this unusual efficiency, it is worth further research to find other applications for scan-first search.

We conclude with two open problems concerning graph connectivity. The first is whether the k-vertex connectivity of an n-vertex *directed* graph can be tested in $k^{O(1)}n^2$ sequential time. The second is whether a sparse certificate for the k-vertex connectivity of an n-vertex undirected graph can be found *deterministically* in time polylogarithmic *both* in k and n, using a number of processors polynomial both in k and n.

Acknowledgements. The authors wish to thank Martin Farach, Hillel Gazit, Samir Khuller, John Reif, Sandeep Sen, and Éva Tardos for helpful discussions.

REFERENCES

[1] A. AGGARWAL AND R. J. ANDERSON, *A random NC algorithm for depth first search*, Combinatorica, 8 (1988), pp. 1–12.

[2] A. AGGARWAL, R. J. ANDERSON, AND M. Y. KAO, *Parallel depth-first search in general directed graphs*, SIAM J. Comput., 19 (1990), pp. 397–409. Also appeared in the Proceedings of the 21st ACM Symposium on Theory of Computing, Seattle, Washington, May 15-17, 1989, pages 297-308.

[3] B. AWERBUCH AND D. PELEG, *Network synchronization with polylogarithmic overhead*, in Proceedings of the 31th Annual IEEE Symposium on Foundations of Computer Science, 1990, pp. 514–522.

[4] C. BERGE, *Graphs*, North-Holland, New York, second revised ed., 1985.

[5] B. BOLLOBÁS, *Extremal Graph Theory*, Academic Press, London, 1978.

[6] A. BORODIN, J. VON ZUR GATHEN, AND J. HOPCROFT, *Fast parallel matrix and gcd computations*, in Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, 1982, pp. 65–71.

[7] J. CHERIYAN AND S. N. MAHESHWARI, *Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs*, Journal of Algorithms, 9 (1988), pp. 507–537.

[8] J. CHERIYAN AND R. THURIMELLA, *Finding sparse spanning subgraphs efficiently.* Preprint, Aug. 1990.

[9] ——, *On determining vertex connectivity*, Technical Report UMIACS-TR-90-79 CS-TR-2485, University of Maryland, College Park, Maryland, June 1990.

[10] ——, *Algorithms for parallel k-vertex connectivity and sparse certificates*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, 1991.

[11] R. COLE AND U. VISHKIN, *Approximate and exact parallel scheduling with applications to list, tree and graph problems*, in Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 478–491.

[12] T. H. CORMEN, C. L. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, The MIT Press, 1990.

[13] S. EVEN, *An algorithm for determining whether the connectivity of a graph is at least k*, SIAM J. Comput., 4 (1975), pp. 393–396.

[14] ——, *Graph Algorithms*, Computer Science Press, 1979.

[15] D. FUSSELL, V. RAMACHANDRAN, AND R. THURIMELLA, *Finding triconnected components by local replacements*, in Proceedings of the 16th International Colloquium on Automata, Languages, and Programming, 1989.

[16] D. FUSSELL AND R. THURIMELLA, *Successive approximation in parallel graph algorithms*, Theoretical Comput. Sci., 74 (1990), pp. 19–35.

[17] Z. GALIL, *Finding the vertex connectivity of graphs*, SIAM J. Comput., 9 (1980), pp. 197–199.

[18] H. GAZIT, *An optimal randomized parallel algorithm for finding connected components in a graph*, Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, (1986), pp. 492 - 501.

[19] H. GAZIT AND G. L. MILLER, *An improved parallel algorithm that computes the BFS numbering of a directed graph*, Information Processing Letters, 28 (1988), pp. 61–65.

[20] V. HADZILACOS, *Connectivity requirements for byzantine agreement under restricted types of failures*, Distributed Computing, 2 (1987), pp. 95–103.

[21] A. ITAI AND M. RODEH, *The multi-tree approach to reliability in distributed networks*, Information and Computation, 79 (1988), pp. 43–59.

[22] A. KANEVSKY AND V. RAMACHANDRAN, *Improved algorithms for graph four-connectivity*, Journal of Computer and System Sciences, 42 (1991), pp. 288–306.

[23] R. KARP AND V. RAMACHANDRAN, *A survey of parallel algorithms for shared-memory machines*, Tech. Rep. UCB/CSD 88/408, Computer Science Division, EECS, University of California at Berkeley, Mar. 1988. To appear in the Handbook of Theoretical Computer Science by North-Holland.

[24] S. KHULLER AND B. SCHIEBER, *Efficient parallel algorithms for testing connectivity and finding disjoint s-t paths in graphs*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 288–293.

[25] K. MULMULEY, U. V. VAZIRANI, AND U. V. VAZIRANI, *Matching is as easy as matrix inversion*, Combinatorica, 7 (1987), pp. 105–114.

[26] H. NAGAMOCHI AND T. IBARAKI, *Linear time algorithms for finding k-edge-connected and k-node-connected spanning subgraphs*, Technical Report #89006, Dept. of Applied Mathematics & Physics, Faculty of Engineering, Kyoto University, 1989. Also, to appear in *Algorithmica*.

[27] R. TARJAN, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

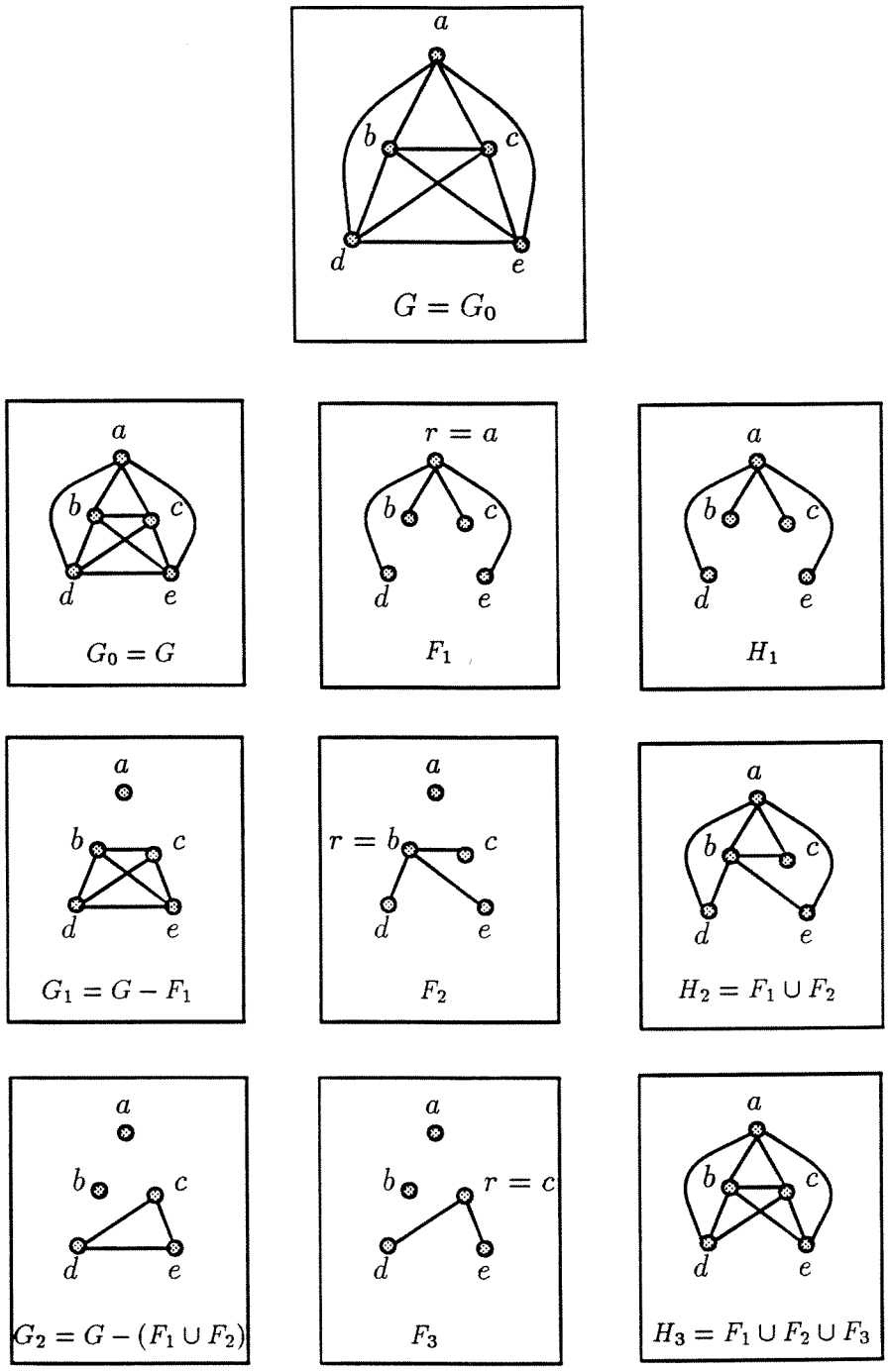

FIG. 4. *An example illustrating the definitions of G_i, F_i, and H_i when G is K_5.*

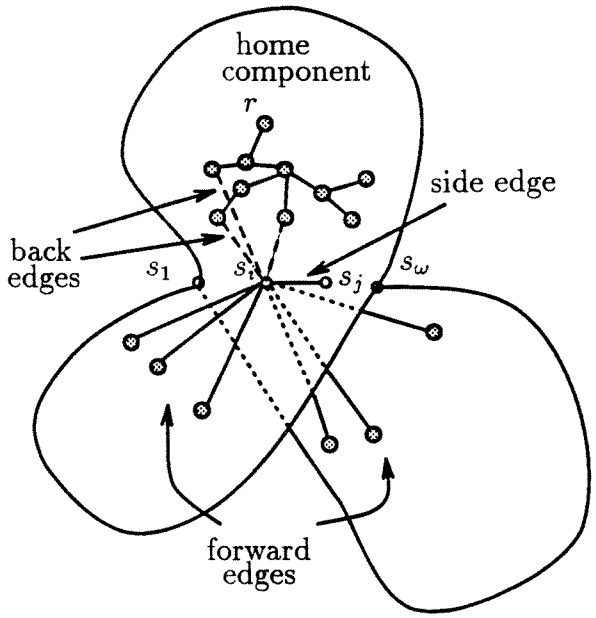

FIG. 5. *Illustration of definitions of home components, and of forward, back, and side edges (of vertex s_i).*

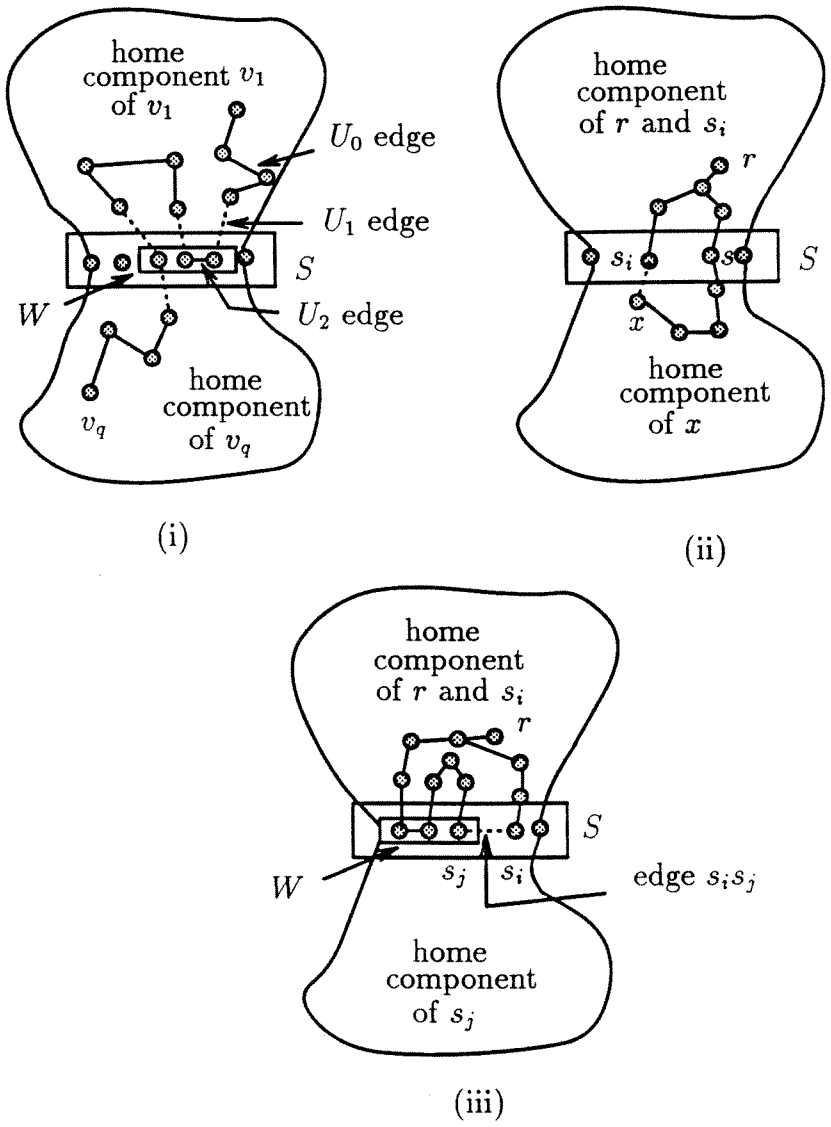

FIG. 6. *Proof of the main certificate theorem. (i) Proof of Claim 3 (in Lemma 2.14) and Lemma 2.15. (ii) Proof of Lemma 2.13 and Claim 4 (in Lemma 2.14). (iii) Proof of Claim 5 (in Lemma 2.14).*

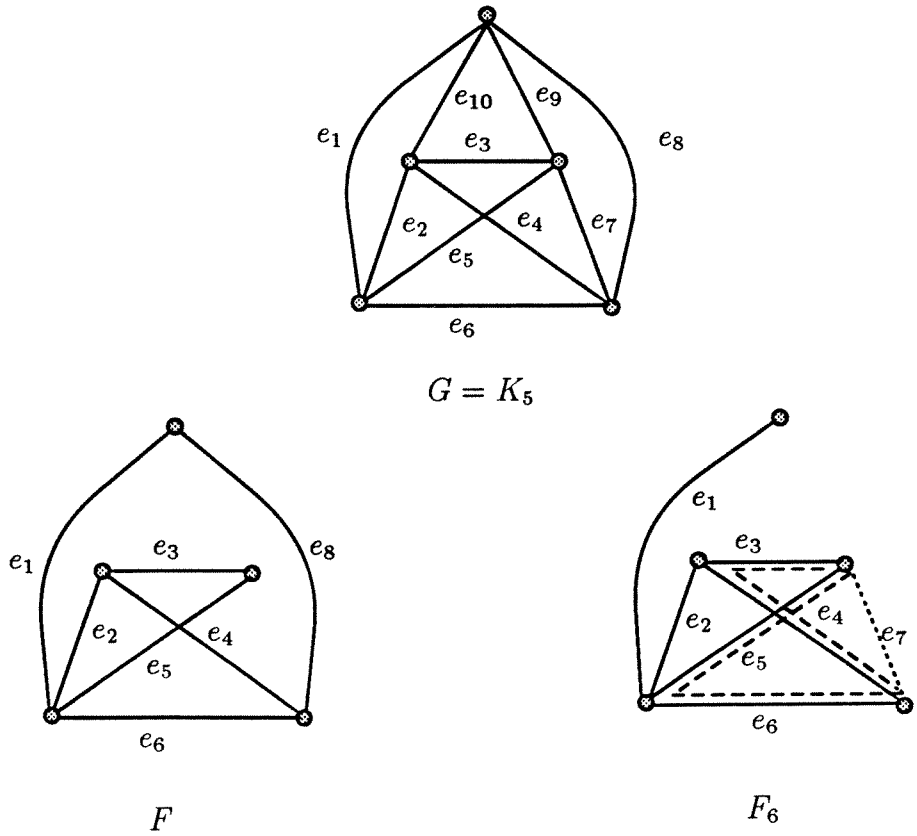

FIG. 7. *Using the online algorithm to find a certificate for 2-vertex connectivity. The edge ordering is* $e_1, e_2, \cdots, e_{10}$. *F does not contain* e_7, *for example, because in* $F_6 = \{e_1, e_2, \cdots, e_6\}$ *there are two vertex-disjoint paths between the endpoints of* e_7.