On Computing the Connectivities of Graphs and Digraphs

Abdol H. Esfahanian
Computer Science Department, Michigan State University, East Lansing, Michigan 48824

S. Louis Hakimi
Electrical Engineering and Computer Science Department, Northwestern University, Evanston, Illinois 60201

In this paper methods are described that will compute the edge-connectivity of a graph or a digraph at least twice as fast as the known methods. A study of the computation of the vertex-connectivity is presented which leads to new algorithms for this purpose or for the purpose of determining if the vertex-connectivity is at least k. These algorithms compare favorably with Kleitman's, Even's, Even and Tarjan's, and Galil's algorithms.

I. INTRODUCTION

All known algorithms for the computation of edge- and vertex-connectivities of graphs and digraphs are based on a number of "calls" to the max-flow algorithm (MFA) [1, 3, 4, 7, 8]. The major part of the computation in such algorithms is due to these calls. Each call terminates as a certain number of flow augmenting paths (FAPs) are found. A natural way of evaluating such algorithms would be to compare the number of calls made to the MFA by each algorithm. The best known algorithm for computing the edge-connectivity of a graph (a digraph) requires $n - 1$ (n) calls to the MFA where n is the number of vertices in the graph [1, 8].

The literature on the computation of the vertex-connectivity is more extensive. Kleitman's procedure for establishing that the vertex-connectivity is at least k requires $k(n - \delta) - \frac{1}{2}k(k + 1)$ calls to the MFA where δ is the minimum vertex degree of G [7]. Even's procedure for the same purpose requires $n - k + \frac{1}{2}k(k - 1)$ calls to the MFA [1]. Even and Tarjan's method for computing the vertex-connectivity κ of a graph requires $(\kappa + 1)(n - \delta - 1) - \frac{1}{2}\kappa(\kappa + 1)$ calls to the MFA [3].

In this paper we first present techniques for the computation of the edge-connectivity of a graph or a digraph which requires at most $\frac{1}{2}n$ "calls" to the MFA. We then describe two simple methods. One is for determining whether the vertex-connectivity is at least k, which requires $n - k + \frac{1}{2}(k - 1)(k - 2)$ calls to the MFA. The other

NETWORKS, Vol. 14 (1984) 355-366

CCC 0028-3045/84/020355-12$04.00

is for computing the vertex-connectivity κ of a graph, which requires $n - \delta - 1 + \frac{1}{2}\kappa(2\delta - \kappa - 3)$ calls to MFA.

Let FAP be a routine which finds (if exists) a FAP from a given source to a given terminus. One may evaluate the algorithms for the computation of connectivities of graphs and digraphs, based on the number of calls to FAP required by each such algorithms, along with the complexity of the FAP routine when applied to the graphs in such algorithms.

Our evaluation in this regard is with the assumption that the complexity of the FAP routine is equal to the number of edges of the network in which the FAP is found. Let "cost function" $C(i)$ of algorithm i be expressed as $C(i) = \Sigma_j c(j)$ where $c(j)$ is the number of edges of the network in which the jth FAP is found.

We will compute the cost function of every one of the algorithms mentioned, including the new implementation of Even's algorithm by Galil [4]. It will be seen that our methods both for determining if the vertex-connectivity is at least k, and for the computation of the vertex-connectivity have smaller costs than the known algorithms in each regard.

II. COMPUTING THE EDGE-CONNECTIVITY OF A GRAPH OR A DIGRAPH

A. Preliminary Observations

We will follow Even's notation closely [2]. Let $G(V, E)$ represent a graph (digraph) without loops or multiple edges with the vertex-set V and the edge-set E. In a graph G, the degree $d(v)$ of vertex $v \in V$ is defined as the number of edges incident at vertex v, whereas in a digraph G, the in-degree $d_{\text{in}}(v)$ and the out-degree $d_{\text{out}}(v)$ of vertex $v \in V$ are defined respectively as the number of edges incoming to and edges outgoing from vertex v. Let u and v be distinct vertices in V. A $u - v$ edge separator is a set of edges $S \subseteq E$, such that every path (directed path in case of digraphs) from the vertex u to the vertex v uses at least one edge of S. $M(u - v)$ represents the least cardinality of a $u - v$ edge separator. The edge-connectivity λ of a graph is defined as $\lambda = \min\{M(u - v)$: unordered pairs $u, v \in V\}$, and λ of a digraph is defined as $\lambda = \min\{M(u - v)$: $u - v \in V \times V\}$. It is known that $M(u - v)$ is equal to max-flow from u to v in a type 1 network of $G(V, E)$, where the network is obtained by setting the capacities of all edges to one [2].

Define S, L, and R as follows: Let S be a minimum edge separator in G, i.e., $|S| = \lambda$. Thus there are at least two vertices u and v such that S is a $u - v$ edge separator. Then let sets L and R be as follows: $L = \{v' \in V$: there is a path (directed path in case of digraphs) from u to v' which avoids $S\}$, and $R = \{v' \in V$: there is no path (directed path in case of digraphs) from u to v' which avoids $S\}$.

Assume S, L, and R are defined as above. Since in a graph $M(u - v) = M(v - u)$ for all u and v in V, for every vertex u in L (or in R), and any vertex v on the other side [i.e., in R (or in L)], $\lambda = M(u - v)$. Thus λ of a graph could be determined if we had an "oracle" as follows:

1. Select a vertex u.
2. Using the "oracle," identify a vertex v on the other side of S.
3. Compute $M(u - v)$, and set $\lambda \leftarrow M(u - v)$; stop.

In the known algorithms [2, 8], u having been selected, then v could be identified to within a set $V - \{u\}$ of vertices and consequently λ is found by $\lambda = \min\{M(u - v): v \in V - \{u\}\}$, which implies $n - 1$ calls to the MFA. We will show that v could be identified to within a set of at most $\frac{1}{2}n$ vertices, provided that $\lambda < \delta$, where $\delta = \min\{d(v): v \in V\}$ in G.

In digraphs $M(u - v)$ is not necessarily equal to $M(v - u)$ for u and v in V. Selecting an arbitrary vertex u and computing $M(u - v)$ for a vertex v on the side that does not contain u is not sufficient, as to get the proper result u must belong to L. Thus λ of a digraph is determined as $\lambda = \min\{M(u - v), M(v - u): v \in V - \{u\}\}$, which implies $2(n - 1)$ calls to the MFA [2]. But the following interesting observation by Schnorr reduces the number of "calls" to the MFA to n [2, 8].

Lemma 1 [2, 8]. Let $u_0, u_1, u_2, \ldots, u_p$ be a circular ordering (i.e., $u_0 = u_p$) of some vertices of G, such that $u_i \in L$ and $u_j \in R$ for some i and j. Then $\lambda = \min\{M(u_i - u_{i+1}): i = 0, 1, 2, \ldots, p - 1\}$.

Clearly, a circular ordering of size n will contain at least one vertex in L and one vertex in R. Thus λ of a digraph could be determined using the above relation, which requires n calls to the MFA. We will show that it is possible to find a circular ordering of size not greater than $\frac{1}{2}n$ which contains at least one vertex in L and one vertex in R, provided that $\lambda < \delta'$, where $\delta' = \min\{\min d_{\text{in}}(v), \min d_{\text{out}}(v)\}$ in G.

B. Computing the Edge-Connectivity of a Graph

Let $G(V, E)$ be a graph; let S, L, and R be defined as above; and assume that $\lambda < \delta$ for lemmas and theorems in this section. Furthermore, throughout this paper, let $A(u)$ denote the set of vertices adjacent with u and $I(u)$ denote the set of edges incident at u. For any $V' \subset V$ and $E' \subset E$, $\langle V' \rangle$ and $\langle E' \rangle$ denote the subgraphs of G induced by V' and E', respectively, and let $n = |V|$ and $e = |E|$.

Lemma 2. $|L| > \delta$ and $|R| > \delta$.

Proof. Suppose $L = \{u_1, u_2, \ldots, u_p\}$ and $d(u_i)$ is the degree of vertex u_i in G. We know $\sum_{i=1}^{p} d(u_i) \geq p\delta$. But $\sum_{i=1}^{p} d(u_i) = 2|E(\langle L \rangle)| + |S|$, where $E(\langle L \rangle)$ represents the edge-set of subgraph $\langle L \rangle$. Thus $\sum_{i=1}^{p} d(u_i) \leq 2(\frac{1}{2}p(p - 1) + |S|$ or $p\delta < p(p - 1) + \delta$, which if $p - 1 > 0$ implies $p > \delta$. However, we know $p > 1$, because $\lambda < \delta$; thus $|L| > \delta$. By the same token $|R| > \delta$.

Lemma 3. For each vertex u in L, let $x(u)$ denote the number of edges in $\langle L \rangle$ incident at u, and $y(u)$ denote the number of edges in S which are not incident at u. Then $x(u) > y(u)$.

Proof. Define $z(u)$ as the number of edges in S that are incident at u. Clearly $z(u) + y(u) = |S| < \delta$ and $z(u) + x(u) = d(u) \geq \delta$. Thus $x(u) > y(u)$. Exactly the same result holds for u in R.

Lemma 4. Let T be a spanning tree of G. Then both L and R will contain at least one vertex which is not a leaf of T.

Proof. We will prove the lemma only for L. Proof for R follows the same lines. Assume otherwise. Thus all vertices in L are leaves of T, which means $|L| = |S|$. Since $|L| > \delta$, this implies $|S| > \delta$, a contradiction as we know $|S| < \delta$.

In the light of the above lemmas, λ can be determined by the following algorithm.

Algorithm 1. Computing λ of graph G.

1. Select a spanning tree T of G.
2. Select a vertex u which is not a leaf of T and let $c = \min\{M(u - v): v \in NL - \{u\}\}$, where $NL = \{v: v \text{ is not a leaf of } T\}$.
3. $\lambda = \min(\delta, c)$; stop.

Clearly Algorithm 1 finds λ of graph G correctly. The more leaves T has, the fewer calls to the MFA Algorithm 1 requires. However, it is known that to find a spanning tree of a graph G with the maximum number of leaves is NP-hard [5]. We will take a different approach and find a spanning tree T' of G such that both L and R contain at least one vertex which is a leaf of T'. This would immediately imply by Lemma 4 that both L and R have leaves as well as nonleaves of T'. The tree T' is constructed using the following algorithm. In this algorithm by $g_1 \cup g_2$ of two subgraphs g_1 and g_2 of $G(V, E)$, we mean a graph whose edge-set and vertex-set are the union of the edge-sets and the vertex-sets of g_1 and g_2, respectively.

Algorithm 2.

1. $T' \leftarrow \phi$.
2. Select a vertex u and set $T' \leftarrow T' \cup \langle I(u) \rangle$.
3. Select a leaf w in T' such that $|A(w) \cap [V - V(T')]| \geqslant |A(r) \cap [V - V(T')]|$ for all leaves r in T'.
4. $T' \leftarrow T' \cup \langle w \cup \{A(w) \cap [V - V(T')]\} \rangle$.
5. If $E(T') < n - 1$ go to Step 3; else stop.

Lemma 5. Let T' be a spanning tree of G constructed by Algorithm 2. Both L and R will contain at least one vertex which is a leaf of T'.

Proof. We shall prove the lemma for L; the proof for R is identical and thus omitted. In the process of constructing T', let v be the first vertex in L selected in Step 3 of Algorithm 2 (the proof would still hold if $v = u$, the vertex selected at Step 2). Let $X(v) = A(v) \cap L$. Note that $X(v)$ is not empty because of the assumption $|S| < \delta$. After the execution of Step 4, T' will contain all vertices in set $X(v)$ as leaves; in other words, L will contain at least $|X(v)| = x(v)$ leaves of T'. If L is not to contain any leaf of T' when T' is fully constructed, the number of leaves of T' in L must eventually decrease to zero. For any such decrease, at least one particular edge of S which is not incident at v is used. This implies S must contain at least $x(v)$ edges none of which is incident at v, a contradiction (see Lemma 3).

Lemmas 4 and 5 immediately imply the following theorem.

Theorem 1. Let T' be a spanning tree of G constructed by Algorithm 2. Then both L and R contain at least one nonleaf vertex and one leaf vertex of T'.

Theorem 1 suggests that λ of a graph G can be determined as follows:

Algorithm 3. Computing λ of graph G.

1. Construct a spanning tree T' of $G(V, E)$ using Algorithm 2.
2. Let $LF(T') = \{v \in V: v \text{ is a leaf of } T'\}$ and $NLF(T') = \{v \in V: v \text{ is not a leaf of } T'\}$ and among these two sets choose the one with smaller cardinality and call it P (clearly $|P| \leqslant \frac{1}{2}n$).
3. Select a vertex $u \in P$ and let $c = \min\{M(u - v): v \in P - \{u\}\}$.
4. $\lambda = \min(\delta, c)$; stop.

Clearly Algorithm 3 determines λ of graph G correctly. Step 3 could be done in $O(\frac{1}{2}\lambda|E||V|)$ operations if we allow concurrency as in [8]. Although Step 1 may take $O(n^2)$ steps, as we require less than $\frac{1}{2}n$ "calls" to the MFA, Algorithm 3 is more efficient than the best known one.

C. Computing the Edge-Connectivity of a Digraph

Let $G(V, E)$ represent a digraph and S, L, and R be defined as above. Furthermore, assume $\lambda < \delta'$ for lemmas and theorem in this section.

Lemma 6. $|L| > \delta'$ and $|R| > \delta'$.

Proof. We know $\Sigma_{v \in L} d_{\text{out}}(v) \geqslant p\delta'$, where $p = |L|$. Also, $\Sigma_{v \in L} d_{\text{out}}(v) = |E(\langle L \rangle)| + |S|$, which implies $p\delta' \leqslant \Sigma_{v \in L} d_{\text{out}}(v) < p(p-1) + \delta'$. As $p > 1$, this immediately implies $|L| = p > \delta'$. The case for $|R|$ follows the same lines.

Lemma 7. For each vertex $u \in L$, define $x'(u)$ as the number of edges in $\langle L \rangle$ which have u as the tail, and define $y'(u)$ as the number of edges in S which do not have u as the tail. Then $x'(u) > y'(u)$,

Proof. Follows the same lines as in Lemma 3. It is important to note that Lemma 7 does not necessarily hold for vertices in R. This is why the methods described so far may not work for digraphs. But there are other spanning trees!

Lemma 8. Let T'' be the rooted tree induced by forward edges produced by a Depth-First Search scanning of G. Then both $\langle L \rangle$ and $\langle R \rangle$ will contain at least one edge of T''.

Proof. In part (a) we will prove the lemma for $\langle R \rangle$ and in part (b) for $\langle L \rangle$.

(a) Assume otherwise, i.e., $\langle R \rangle$ does not contain any edge of T''. Thus each vertex in R must have gotten scanned from some vertex in L. This implies that for each vertex u in R there exists a vertex v in L such that edge (v, u) belongs to S. This makes $|S| = |R| > \delta'$, a contradiction. Thus $\langle R \rangle$ contains at least one edge of T''.

(b) Assume $\langle L \rangle$ does not contain any edge of T'', and let $w \in L$ be a leaf of T'' and further assume that w is the first leaf of T'' obtained in the process of scanning $G(V, E)$. Define $X' = \{v \in V:$ edge (w, v) belongs to the edge-set of $\langle L \rangle\}$. For w to be a leaf of T'', all vertices in set X' must have gotten scanned before w. Since w is the first leaf, none of the vertices in set X' is a leaf. Thus there are $|X'| = x'(w)$ edges which belong

to S of which w is not a tail, a contradiction to Lemma 7. Thus all vertices in L are nonleaves of T''. This implies $|S| = |L| > \delta'$, a contradiction. Therefore $\langle L \rangle$ contains at least one edge of T''.

Theorem 2. The set composed of vertices at the even (odd) levels of T'' contains at least two vertices of which one is in L and one is in R.

Proof. Since the endvertices of an edge in a tree are at levels of different parity and by Lemma 8, $\langle L \rangle$ and $\langle R \rangle$ contain at least one edge of T'', thus the theorem.

Combining Theorem 2 and Schnorr's lemma we have the following algorithm, which determines λ of a digraph G.

Algorithm 4. Computing λ of a digraph G.

1. Construct tree T'' for $G(V, E)$.
2. Let $\text{EVEN}(T'') = \{v \in V: v \text{ is at an even level of } T''\}$ and $\text{ODD}(T'') = \{v \in V: v \text{ is at an odd level of } T''\}$ and between these two sets choose the one with smaller cardinality and call it P (clearly $|P| \leqslant \frac{1}{2}n$) and assume $p = |P|$.
3. Form a circular ordering $U = \{u_0, u_1, u_2, \ldots, u_p\}$ (with $u_0 = u_p$) of vertices in set P and compute $c' = \min\{M(u_i - u_{i+1}): i = 0, 1, 2, \ldots, p - 1\}$.
4. $\lambda = \min(\delta', c')$; stop.

Clearly Algorithm 4 finds λ of digraph G correctly and no more than $\frac{1}{2}n$ calls are made to the MFA. Other comments are the same as for Algorithm 3.

Although Algorithm 4 could be used for graphs, we have found, in almost all cases, Algorithm 3 superior for graphs. This is because in the construction of tree T'' one does not exercise much control over the size of $\text{EVEN}(T'')$ and $\text{ODD}(T'')$, while in construction of tree T' for use in Algorithm 3 one creates a tree with large number of leaves.

III. COMPUTING THE VERTEX-CONNECTIVITY OF A GRAPH

A. Preliminary Observations

Let $G(V, E)$ represent a graph without loops or multiple edges with the vertex-set V and edge-set E. A $u - v$ vertex separator is a set of vertices, $S' \subseteq V - \{u, v\}$ such that every path from u to v uses at least one vertex of S'. Define $N(u - v)$ as follows: If edge $(u, v) \in E$ then $N(u - v) = n - 1$, otherwise $N(u - v)$ is the least cardinality of a $u - v$ vertex separator [2]. The vertex-connectivity $\kappa(G)$ of graph G is defined as $\kappa(G) = \min\{N(u - v)$: unordered pairs $u, v \in V\}$. It is known that $N(u - v)$ is equal to the max-flow from u to v in a network obtained from G where the network is a weighted digraph with $2n$ vertices and $n + 2e$ edges and the capacities of the edges are "1" [2].

Define S', $L(S')$, and $R(S')$ as follows: Let S' be a minimum vertex separator in G, i.e., $|S'| = \kappa(G)$. Thus there are at least two vertices u and v such that S' is a $u - v$ vertex separator. Let $L(S') = \{v' \in V: v'$ is not in S' and there is a path from u to v' which avoids $S'\}$, and let $R(S') = \{v' \in V: v'$ is not in S' and there is no path from u to v' which avoids $S'\}$. We may refer to $L(S')$ and $R(S')$ as the "sides" of S'.

Assume G, $\kappa(G)$ (or simply κ when G is known from the context), S', $L(S')$, and $R(S')$ are defined as above. Clearly for any vertex u in $L(S')$ [or in $R(S')$] and any vertex v in $R(S')$ [or in $L(S')$], $N(u - v) = \kappa$. This implies that in order to compute κ, selecting an arbitrary vertex u and computing $\min\{N(u - v): v \in V\}$ may not be sufficient, because for this to be the proper result G must have a minimum vertex separator which does not contain vertex u. But if we take a set $P \subset V$ such that $|P| > \kappa$, for every minimum vertex separator S' in G, there exists at least one vertex in P which is not in S' and κ could be computed as $\kappa = \min\{\min\{N(u_i - v): v \in V\}: u_i \in P\}$. This is the main idea in Even and Tarjan's algorithm [3].

B. Computing the Vertex-Connectivity of a Graph

The following algorithm is due to Even and Tarjan [3].

Algorithm 5. Computing κ of graph $G(V, E)$.

1. $i \leftarrow 1, N \leftarrow (n - 1)$, and let $V = \{u_1, u_2, \ldots, u_n\}$.
2. For each $j, j = i + 1, i + 2, \ldots, n$,
   (a) if $i > N$ go to Step 4;
   (b) if edge (u_i, u_j) is not in E, then compute $N(u_i - u_j)$, and set $N \leftarrow \min\{N, N(u_i - u_j)\}$.
3. $i \leftarrow i + 1$, go to Step 2.
4. $\kappa(G) \leftarrow N$; stop.

If the MFA is used for the computation of $N(u_i - u_j)$, Algorithm 5 requires $MC(5) = (\kappa + 1)(n - \delta - 1) - \frac{1}{2}\kappa(\kappa + 1)$ calls, provided that G contains an independent set of size $\kappa + 1$ [6]. If FAP is used for the computation of $N(u_i - u_j)$, Algorithm 5 requires $FC(5) = \delta MC(5) - (\delta - \kappa)$ calls to FAP with complexity $n + 2e$. Thus $C(5) = (n + 2e)FC(5)$. However, the following observation would lead to a more efficient way of computing κ.

For an arbitrary vertex $r \in V$, there are two cases that could happen in regard to the relation of r with minimum vertex separators in G.

(i) The graph G has a minimum vertex separator S' which does not contain r. This implies that for any vertex v on the other "side" of S' where r is not, $\kappa(G) = N(r - v)$. Clearly vertex v could be identified to within a set of $V - A(r) - \{r\}$ vertices [in the final section of this paper we suggest a procedure by which v could be identified to within a set of smaller size than $V - A(r) - \{r\}$]. Thus, in this case, $\kappa(G)$ could be computed as $\kappa(G) = \min\{N(r - v): v \in V - A(r) - \{r\}\}$.

(ii) The vertex r is in all minimum vertex separators in G. Let S' be one of these minimum vertex separators and $L(S')$ and $R(S')$ be defined as above. Obviously each "side" of S' contains at least one vertex of set $A(r)$. Actually we claim that each "side" of S' would contain at least two vertices of set $A(r)$ provided each "side" contains more than one vertex. This is because if, for example, $L(S')$ contains only one vertex r' of set $A(r)$, then the set $\{r'\} \cup (S' - \{r\})$ would be a minimum vertex separator in G which does not contain r, contrary to our assumption. Note that if, say, $L(S') = \{x\}$, this is of no concern, since in this case $d(x) = \delta$ and $\kappa(G) = \delta$. In particular, if we select r to be a vertex of minimum degree, then $A(r)$ is also a minimum ver-

tex separator in G and does not contain r. Thus if we take a set $P \subset A(r)$ such that $|P| = |A(r)| - 1$, P will contain at least one vertex in $L(S')$ and one in $R(S')$. This implies that κ could be computed as $\kappa = \min\{N(u - v)$: unordered pairs u, v in P and edge (u, v) not in $E\}$.

Not knowing which of the above cases is true for an arbitrary vertex u, both cases must be considered, hence the following algorithm.

Algorithm 6. Computing κ of graph $G(V, E)$.

1. $i \leftarrow 1$, $N \leftarrow (n - 1)$, select a vertex $u \in V$ of minimum degree, and let $A(u) = \{u_1, u_2, \ldots, u_\delta\}$.
2. For each v in $V - A(u) - \{u\}$, compute $N(u - v)$ and set $N \leftarrow \min\{N, N(u - v)\}$.
3. For each $j, j = i + 1, i + 2, \ldots, \delta - 1$,
   (a) if $i \geq \delta - 2$ or $i \geq N$ go to Step 5;
   (b) if edge (u_i, u_j) is not in E, then compute $N(u_i - u_j)$, and set $N \leftarrow \min\{N, N(u_i - u_j)\}$.
4. $i \leftarrow i + 1$, go to Step 3.
5. $\kappa(G) \leftarrow N$; stop.

It should not be difficult to see that Algorithm 6 computes $\kappa(G)$ correctly. If the MFA is used for computation of $N(u_i - u_j)$, Step 2 requires $n - \delta - 1$ calls and Step 3 requires $\frac{1}{2}\kappa(2\delta - \kappa - 3)$ calls. Thus Algorithm 6 requires $MC(6) = n - \delta - 1 + \frac{1}{2}\kappa(2\delta - \kappa - 3)$ calls to the MFA, which is substantially less than the number of calls required by Algorithm 5. If FAP is used for the computation of $N(u_i - u_j)$, Algorithm 6 requires $FC(6) = \delta MC(6) - (\delta - \kappa)$ calls where the complexity of FAP is equal to $n + 2e$. Thus $C(6) = (n + 2e)FC(6)$.

C. Determining Whether a Graph Is k-Connected

If $\kappa(G) \geq k$ we say G is k-connected. To establish that G is k-connected, Kleitman uses the fact that if vertex u belongs to a minimum vertex separator in G, then $\kappa(G) = \kappa(G - \{u\}) + 1$ [7]. He then proceeds as follows.

Algorithm 7. Determining whether $G(V, E)$ is k-connected [7].

1. Let $V = \{u_1, u_2, \ldots, u_n\}$; G is k-connected if none of the tests in the following steps fails.
2. Select a vertex u_1 and check whether $N(u_1 - v) \geq k$ for each v in G.
3. Select a vertex u_2 and check whether $N(u_2 - v) \geq k - 1$ for each v in $G - \{u_1\}$.
4. Select a vertex u_3 and check whether $N(u_3 - v) \geq k - 2$ for each v in $G - \{u_1, u_2\}$. . . .

$j + 1$. Select a vertex u_j and check whether $N(u_j - v) \geq k - j - 1$ for each v in $G - \{u_1, u_2, u_3, \ldots, u_{j-1}\}$. . . .

$k + 1$. Select a vertex u_k and check whether $N(u_k - v) \geq 1$ for each v in $G - \{u_1, u_2, u_3, \ldots, u_{k-1}\}$.

If the MFA is used for the computation of $N(u_i - u_j)$, Algorithm 7 requires $MC(7) = k(n - \delta) - \frac{1}{2}k(k + 1)$ calls, provided that $d(u_i) = \delta$ in $G - \{u_1, u_2, \ldots, u_{i-1}\}$ for $i = 2, 3, \ldots, k - 1$. If FAP is used instead, for each i, $i = 2, 3, \ldots, k + 1$, Step i requires

$(k - i + 2)(n - \delta - i + 1)$ calls, with the cost of FAP equal to $n - i + 2 + 2[e - (i - 2)\delta]$. Thus we have $C(7) = \sum_{j=0}^{k} [n - j + 2(e - j\delta)][(k - j)(n - \delta - j - 1)]$.

Let $G'(j)$ be the graph obtained from $G(V, E)$ by introducing a new vertex u_0 and connecting it to all vertices in set $W(j)$, where $W(j) \subset V$ and $|W(j)| = j$. Even's procedure for determining if G is k-connected is as follows [1].

Algorithm 8. Determining whether $G(V, E)$ is k-connected [1].

1. Let $V = \{u_1, u_2, \ldots, u_n\}$ and $V(k) = \{u_1, u_2, \ldots, u_k\}$; G is k-connected if none of the tests in the following steps fails.
2. For each unordered pair u_i and u_j in $V(k)$ check whether $N(u_i - u_j) \geqslant k$ in G.
3. For each $j, j = k, k + 1, \ldots, n - 1$, let $W(j) = \{u_1, u_2, \ldots, u_j\}$ and form $G'(j)$ and check whether $N(u_0 - u_{j+1}) \geqslant k$.

It should be noted here, as Galil also points out in [4], that Algorithm 8 would also work if $G'(j)$ is replaced by $G'(k)$ in Step 3. If MFA is used, Algorithm 8 requires $n - k + \frac{1}{2}k(k - 1)$ calls. If FAP is used instead, Step 2 requires $\frac{1}{2}k^2(k - 1)$ calls, with the cost of each call equal to $n + 2e$ and for each $j, j = k + 1, k + 2, \ldots, n$ Step 3 makes k calls of cost $n + 2e + j - 1$. Thus we have $C(8) = \frac{1}{2}(n + 2e)k^2(k - 1) + \sum_{j=k+1}^{n} [(n + 2e + j - 1)k]$.

A mixture of Even's idea and our idea for computing the vertex-connectivity have led us to the following algorithm, which requires fewer calls to the MFA than Even's. But before we proceed we present the following remark and lemma.

Let S' be a minimum vertex separator in G; let $L(S')$ and $R(S')$ be defined as before; and let vertices r_1, r_2, and r_3 be in S', $L(S')$, and $R(S')$, respectively. Define $P_1 \subseteq [V - A(r_1)]$, $P_2 \subseteq [L(S') \cup S'] - A(r_2)$, and $P_3 \subseteq [R(S') \cup S')] - A(r_3)$. Furthermore, let, for $i = 1, 2$, and 3, $G(P_i)$ be the graph obtained from G by connecting r_i to every vertex in the set P_i. It should not be difficult to see that S' is also a minimum vertex separator in $G(P_i)$, for $i = 1, 2, 3$.

Lemma 9. Let S' be a minimum vertex separator in $G(V, E)$ and $L(S')$ and $R(S')$ be defined as before and u be a vertex in S'. Define $A_1 = A(u) \cap L(S')$, $A_2 \subseteq A(u) \cap [S' \cup R(S')]$, and $A' = A_1 \cup A_2$. Furthermore, let G'' be the graph obtained from G by removing all the edges in set $E' = \{(u, v') \in E: v' \in A'\}$. Then $\kappa(G'') = \min\{N(u - v'): v' \in A'\}$.

Proof. Clearly for $v_1 \in A_1$, $S' - \{u\}$ is a $u - v_1$ vertex separator in G''. Thus $\kappa(G'') \leqslant \kappa(G) - 1$. Let S'' be an arbitrary minimum vertex separator in G''. This implies that $|S''| = \kappa(G'') \leqslant \kappa(G) - 1$. The vertex u must not belong to S'', for otherwise, the edge-set E' could be restored, leaving S'' as minimum vertex separator in G (see above remark). Also, without loss of generality, if u is in $L(S'')$, $R(S'')$ must contain at least one vertex $v' \in A'$, for otherwise, again the edge-set E' could be restored, leaving S'' as minimum vertex separator in G. Thus $\kappa(G'') = \min\{N(u - v'): v' \in A'\}$.

Algorithm 9. Determining whether $G(V, E)$ is k-connected.

1. Select a vertex u in V of minimum degree, and let $A(u) = \{u_1, u_2, \ldots, u_\delta\}$ and $A(k - 1) = \{u_1, u_2, \ldots, u_{k-1}\}$; G is k-connected if none of the tests in the following steps fails.

2. Check whether $N(u - v) \geq k$, for each v in $V - A(u) - \{u\}$.
3. Check whether $N(u_i - u_j) \geq k$ for every unordered pair u_i and u_j in $A(k-1)$.
4. Form G'' by removing from G all edges in the set $E' = \{(u, v) \in E: v \in [A(u) - A(k-1)]\}$, and for each i, $i = k, k+1, \ldots, \delta$ check whether $N(u - u_i) \geq k - 1$ in G''.

Clearly if G is k-connected none of the tests in Steps 2–4 would fail. If G is not k-connected and Steps 2 and 3 do not fail, then $A(k-1) \subset S' \cup L(S')$ or $A(k-1) \subset S' \cup R(S')$, where S' is an arbitrary minimum vertex separator in G. This is because otherwise $A(k-1)$ will contain a vertex u_i in $L(S')$ and a vertex $u_j \in R(S')$ and $N(u_i - u_j) < k$, which implies that a test in Step 3 should have failed. However, if $A(k-1) \subset S' \cup R(S')$, then by Lemma 9, $\kappa(G'') = \min\{N(u, v'): v' \in [A(u) - A(k-1)]\} < k - 1$. Hence Step 4 would fail. This concludes the validity of Algorithm 9.

If the MFA is used for computation of $N(u_i - u_j)$, Algorithm 9 requires $MC(9) = n - k + \frac{1}{2}(k-1)(k-2)$ calls, which is less than the number of calls to the MFA required by Algorithm 8. If FAP is used instead, Step 2 requires $k(n - \delta - 1)$ calls each of cost $n + 2e$, Step 3 requires $\frac{1}{2}k(k-1)(k-2)$ calls each of cost $n + 2e$, and finally Step 4 requires $(k-1)(\delta - k + 1)$ calls each of cost $n + 2(e - k)$. Thus $C(9) = k(n + 2e)[n - \delta - 1 + \frac{1}{2}(k-1)(k-2)] + (k-1)(\delta - k + 1)[n + 2(e - k)]$, which is smaller than $C(8)$.

D. Further Refinements

In this section, we will present some ideas that almost always reduce the amount of computation for some of the algorithms presented in this section of the paper.

(i) Regarding Algorithm 6, for the case that a vertex u is not in the minimum vertex separator S' in $G(V, E)$, a vertex v on the "side" of S' where u is not is identified to within a set of $V - A(u) - \{u\}$ vertices. We have found the following procedure to be very effective in minimizing the size of the set of vertices to within which v could be identified: Replace ">" by "<" in Step 3 of Algorithm 2 and construct a spanning tree T of G beginning with vertex u according to this new algorithm. Let the set P be formed as follows: For each "father" w in T, ignore arbitrarily one of its "sons" and include the rest of them in P. Of course, the vertex u and the vertices adjacent with u are not included in P.

Lemma 10. For graph G, select a minimum degree vertex u and let T and then P be constructed as above. Then $\min\{N(u - v): v \in V - \{u\}\} = \min\{N(u - v): v \in P\}$, if P is not empty. And if P is empty, then $\min\{N(u - v): v \in V - \{u\}\} = \delta$.

Proof. Let S' be a minimum vertex separator which does not contain u, where minimum is taken only among such separators. Refer to the side of S' that does not contain u as $M(S')$. If S' is such that every vertex in S' is adjacent with at least two vertices in $M(S')$, we are done because T would then contain at least one vertex which is in S', and also is a father of at least two sons in $M(S')$. Thus deletion of one of the sons in the process of forming P would still leave one vertex in P which is also in $M(S')$ and hence $\min\{N(u - v): v \in V - \{u\}\} = \min\{N(u - v): v \in P\}$.

Now suppose every minimum vertex separator S' which does not contain u has a vertex u_1 which is adjacent with exactly one vertex u_2 in $M(S')$. Let us assume that among all such minimum vertex separators, S' is such that $|M(S')|$ is minimum. Observe that if $M(S')$ consisted of a single vertex, say, x, then $d(x) = \delta = d(u) = |S'|$ and $\min\{N(u - v)\colon\ v \in V - \{u\}\} = \min\{N(u - v)\colon\ v \in P\} = \delta$. Suppose, therefore, that $|M(S')| \geqslant 2$. Now it is clear that $S_1 = (S' - \{u_1\}) \cup \{u_2\}$ is a minimum vertex separator which does not contain u. Since $|M(S_1)| < |M(S')|$, this leads to a contradiction; hence the Lemma.

Although the size of the set P as a function of n, δ, and κ is difficult to determine we have found it to be small relative to the size of $V - A(u) - \{u\}$ for many graphs. It can be easily seen that for all nontrivial situations $|P| < |V - A(u)| - 2$.

(ii) Let S' be a minimum vertex separator in G and $L(S')$ and $R(S')$ be defined as before. Clearly $|L(S')| > \delta - |S'|$ and $|R(S')| > \delta - |S'|$. One may use this fact and reduce by $\delta - k$ the number of calls to the MFA required by Algorithms 8 and 9.

(iii) In graph $G(V, E)$, let S' be a minimum vertex separator and $u \in V$ be a vertex such that u is not in S'. Furthermore, let T'' be the rooted tree induced by a DFS scanning of G. It is not difficult to see that both $\langle L(S')\rangle$ and $\langle R(S')\rangle$ will contain an edge of T'' provided that $\kappa \leqslant \frac{1}{2}\delta$. This implies that the set P defined as in Algorithm 4 contains at least a vertex v such that $N(u - v) = |S'| = \kappa$. Thus when using Algorithm 9 with $k \leqslant \frac{1}{2}\delta$, one could replace the set $V - A(u) - \{u\}$ in Step 2 of the Algorithm by the set P defined as above. This would reduce by one-half the number of calls required by the Step 2 of Algorithm 9.

(iv) Both Algorithms 7 (Kleitman's) and 8 (Even's) could be easily modified to give algorithms which compute the vertex-connectivity of a graph. For example the following is a modification of Algorithm 8.

Algorithm 10. Computing κ of $G(V, E)$.

A. $i \leftarrow 1, N \leftarrow (n - 1)$, and let $V = \{u_1, u_2, \ldots, u_n\}$ and $W(\delta) = \{u_1, u_2, \ldots, u_\delta\}$.

B. Form $G'(\delta)$ and for each j, $j = \delta + 1, \delta + 2, \ldots, n$ compute $N(u_0 - u_j)$, and set $N \leftarrow \min\{N, N(u_0 - u_j)\}$.

C. For each j, $j = i + 1, i + 2, \ldots, \delta$

(a) if $i \geqslant N$ go to Step E,

(b) if edge $u_i - u_j$ is not in E, then compute $N(u_i - u_j)$ in G, and set $N \leftarrow \min\{N, N(u_i - u_j)\}$.

D. $i \leftarrow i + 1$, go to Step C.

E. $\kappa(G) \leftarrow N$; stop.

This Algorithm requires $n - \delta + \frac{1}{2}(\kappa + 1)(2\delta - \kappa - 2)$ calls to the MFA, which is less efficient than Algorithm 6. Nevertheless, the above modification of Even's algorithm requires fewer calls to the MFA than Algorithm 5 (Even and Tarjan's) and is also more efficient than finding κ by using Algorithm 8 (Even's) and doing a "careful binary search" as mentioned in [4].

(v) Galil in [4] computes $\kappa(G)$ by allowing concurrency in determining whether G is 1-connected, 2-connected, ..., using Algorithm 8. Although his method requires fewer calls to FAP than Algorithm 6, it uses more space than our algorithm and also

FAPs are found in bigger networks. Consequently the cost of Galil's algorithm, which is equal to the cost of Algorithm 9 when $k = \kappa + 1$, is much higher than $C(6)$.

This work was supported in part by the National Science Foundation under grants ECS-8121741 and ECS-8201387.

References

[1] S. Even, An Algorithm for determining whether the connectivity is at least k. *SIAM J. Computing* 4 (1975) 393–396.
[2] S. Even, *Graph Algorithms.* Computer Science, Polomac, MD (1979) pp. 116–132.
[3] S. Even and R. E. Tarjan, Network flow and testing graph connectivity. *SIAM J. Computing* 4 (1975) 507–518.
[4] Z. Galil, Finding the vertex connectivity of graphs. *SIAM J. Computing* 9 (1980) 197–199.
[5] M. R. Garey and D. S. Johnson, *Computers and Intractibility.* Freeman, San Francisco (1979), p. 206.
[6] F. Harary, *Graph Theory.* Addison-Wesley, Reading, MA (1972), pp. 94–96.
[7] D. J. Kleitman, Methods for investigating connectivity of large graphs. *IEEE Trans. Circuit Theory* CT-16 (1969) 232–233.
[8] C. P. Schnorr, Bottlenecks and edge connectivity in unsymmetrical networks. *SIAM J. Computing* 8 (1979) 265–274.

Received January 31, 1983
Accepted August 22, 1983