Algorithmica (1992) 7: 583–596

Algorithmica

A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph[1]

Hiroshi Nagamochi[2] and Toshihide Ibaraki[2]

Abstract. We show that any k-connected graph $G = (V, E)$ has a sparse k-connected spanning subgraph $G' = (V, E')$ with $|E'| = O(k|V|)$ by presenting an $O(|E|)$-time algorithm to find one such subgraph, where connectivity stands for either edge-connectivity or node-connectivity. By using this algorithm as preprocessing, the time complexities of some graph problems related to connectivity can be improved. For example, the current best time bound $O(\max\{k^2|V|^{1/2}, k|V|\}|E|)$ to determine whether node-connectivity $\kappa(G)$ of a graph $G = (V, E)$ is larger than a given integer k or not can be reduced to $O(\max\{k^3|V|^{3/2}, k^2|V|^2\})$.

Key Words. Undirected graphs, Spanning subgraphs, Connectivity, k-edge-connectivity, k-node-connectivity, Linear-time algorithms.

1. Introduction. In this paper connectivity stands for either edge-connectivity or node-connectivity unless explicitly specified. A graph $G = (V, E)$ stands for an undirected graph that satisfies $|V| \geq 2$. It may have multiple edges but has no self-loop, unless otherwise specified, though a simple graph is always assumed when node-connectivity is discussed. Given a k-connected graph, the problem of finding a k-connected spanning subgraph with the minimum number of edges is known to be NP-complete for any fixed k (≥ 2) [3, Problem GT31]. Recently, Suzuki *et al.* [9] emphasized the importance of a "linear"-time algorithm to find a "sparse" k-connected spanning subgraph of a given k-connected graph. We show that any k-connected graph $G = (V, E)$ has a k-connected spanning subgraph $G' = (V, E')$ with $|E'| = O(k|V|)$. This result was also independently obtained by Nishizeki and Poljak [8]. For special cases of $k = 2$ and 3, Suzuki *et al.* [9] give an $O(|E|)$-time algorithm to find a k-node-connected subgraph $G' = (V, E')$ with $|E'| \leq 3|V| - 5$ for $k = 2$ and $|E'| \leq 3|V| - 3$ for $k = 3$. For general k, Nishizeki and Poljak [8] find a k-edge-connected G' with $|E'| \leq k(|V| - 1)$ in $O(k|E|)$ time, and a k-node-connected G' with $|E'| \leq k(|V| - 1)$ in $O(|V|^{1/2}|E|^2)$ time.

In this paper we present $O(|E|)$-time algorithms for both problems of k-edge-connected subgraphs and k-node-connected subgraphs. The spanning subgraph $G' = (V, E')$ obtained satisfies $|E'| \leq k|V| - k(k + 1)/2$ ($\leq k(|V| - 1)$) (in case of

[1] The first author was partially supported by the Grant-in-Aid for Encouragement of Young Scientists of the Ministry of Education, Science and Culture of Japan and by the subvention to young scientists by the Research Foundation of Electrotechnology of Chubu.

[2] Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University, Kyoto 606, Japan.

Received September 6, 1989; revised April 16, 1990, and August 16, 1990. Communicated by Greg N. Frederickson.

node-connectivity, G is assumed to be simple). By using this algorithm as proprocessing, the time complexity of algorithms for solving other graph problems can be improved, as is discussed in the last section.

2. *k*-Edge-Connected Subgraph. In this section a graph is possibly multiple, but, if there is no confusion, an edge e with the set of end nodes $\{u, v\}$ is denoted by $e = (u, v)$. Our algorithm requires the following property, which was also independently found by Nishizeki and Poljak [8].

LEMMA 2.1. *For a graph $G = (V, E)$, simple or multiple, let $F_i = (V, E_i)$ be a maximal spanning forest in $G - E_1 \cup E_2 \cup \cdots \cup E_{i-1}$, for $i = 1, 2, \ldots, |E|$, where possibly $E_i = E_{i+1} = \cdots = E_{|E|} = \varnothing$ for some i. Then each spanning subgraph $G_i = (V, E_1 \cup E_2 \cup \cdots \cup E_i)$ satisfies*

$$\lambda(x, y; G_i) \geq \min\{\lambda(x, y; G), i\} \qquad \textit{for all} \quad x, y \in V, \tag{2.1}$$

where $\lambda(x, y; H)$ denotes the local edge-connectivity between x and y in graph H.

PROOF. We proceed by induction on i. Equation (2.1) for $i = 1$ is obvious from the maximality of forest F_1. Assume that some $x, y \in V$ and i satisfy $\lambda(x, y; G_i) \geq \min\{\lambda(x, y; G), i\}$ and $\lambda(x, y; G_{i+1}) < \min\{\lambda(x, y; G), i + 1\}$. By $\lambda(x, y; G_i) \leq \lambda(x, y; G_{i+1})$, this means $\lambda(x, y; G) \geq i + 1$ and $\lambda(x, y; G_i) = \lambda(x, y; G_{i+1}) = i$. That is, G_{i+1} has a minimal cut set $W \subseteq E$ with $|W| = i$ such that there are two distinct components X and Y in $G_{i+1} - W$ containing x and y, respectively. Since $|W - E_{i+1}| \geq i$ follows from $\lambda(x, y; G_i) = i$, this W satisfies $W \cap E_{i+1} = \varnothing$. However, there is an edge $e \in E - E_1 \cup E_2 \cup \cdots \cup E_{i+1}$ connecting a node in X to a node in Y, by $\lambda(x, y; G) \geq i + 1$, and this contradicts the maximality of $F_{i+1} = (V, E_{i+1})$. □

By this lemma we see that $G_k = (V, E')$, where $E' = E_1 \cup E_2 \cup \cdots \cup E_k$, is k-edge-connected if $k \leq$ the edge-connectivity $\lambda(G)$. Moreover, $G_k = (V, E')$ satisfies $|E'| \leq k(|V| - 1)$ since $|E_i| \leq |V| - 1$ for all i. It is therefore easy to see that G_k can be obtained in $O(k(|V| + |E|))$ time by repeating graph search procedure k times. However, this time complexity can be reduced to $O(|V| + |E|)$, as discussed below. The idea is to constructed all $E_1, E_2, \ldots, E_{|E|}$ in a single scan. During the graph search we compute, for each e being scanned, the i satisfying $e \in E_i$. Such i can be defined to be the smallest i such that $E_i \cup \{e\}$ does not contain a cycle, where these E_i denote the edge sets constructed so far. Note that, in general, checking whether $E_i \cup \{e\}$ contains a cycle requires $O(|E_i|)$ time. To reduce this to $O(1)$, we always chose an unscanned edge e that is adjacent to an edge $e' \in E_i$ with the largest i. This graph search procedure can be described as follows:

Procedure FOREST; {input: $G(V, E)$, output: $E_1, E_2, \ldots, E_{|E|}$}
{Let $r(v) := i$ denote that v has been reached by an edge of the forest $F_i = (V, E_i)$.}

```
begin
  E_1 := E_2 := ··· := E_|E| := ∅;
  Label all nodes v ∈ V and all edges e ∈ E "unscanned";
  r(v) := 0 for all v ∈ V;
  while there exist "unscanned" nodes do
    begin
      Choose an "unscanned" node x ∈ V with the largest r;
      for each "unscanned" edge e incident to x do
        begin
          E_r(y)+1 := E_r(y)+1 ∪ {e};   {y is the other end node (≠x)
          of e}
          if r(x) = r(y) then r(x) := r(x) + 1;
          r(y) := r(y) + 1;
          Mark e "scanned"
        end;
      Mark x "scanned"
    end;
end.
```

For example, a partition E_i, $i = 1, 2, \ldots$, obtained by applying FOREST to a simple graph G^1 of Figure 1 is shown in Figure 2. Similarly, a partition E_i, $i = 1, \ldots$, of a multiple graph G^2 is illustrated in Figure 3. In Figures 2 and 3 node x_i (edge e_j) represents that this is the ith node (jth edge) scanned by FOREST. The edges are directed from x to y of FOREST, so that the role of these nodes is explicitly shown.

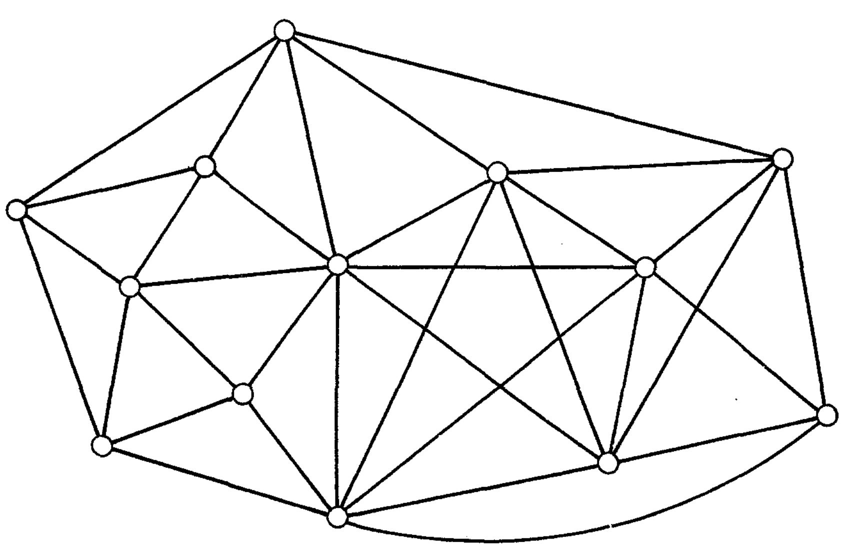

Fig. 1. A simple graph G^1 with $\lambda(G^1) = 4$ and $\kappa(G^1) = 3$.

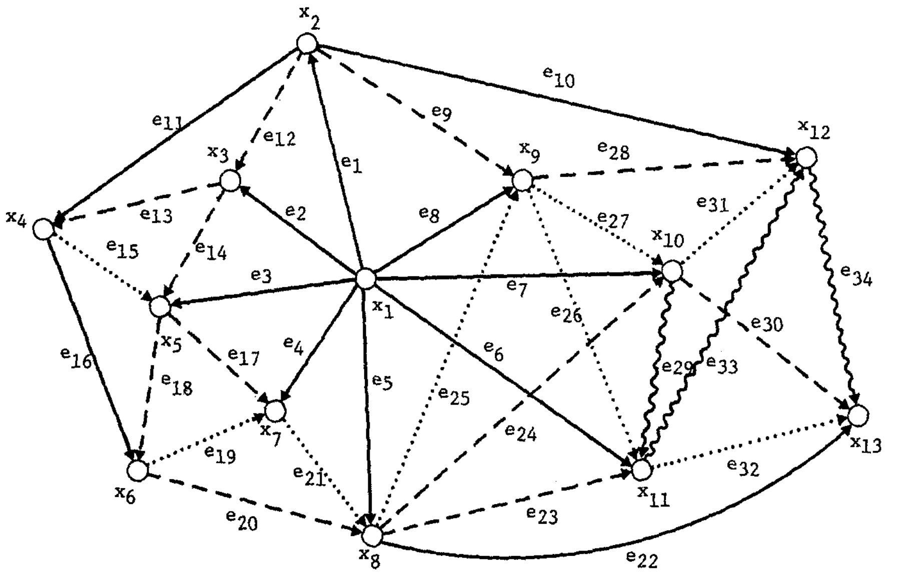

Fig. 2. Partition E_i of G^1 obtained by FOREST. o——o, edges in E_1; o– – –o, edges in E_2; o·········o, edges in E_3; o∿∿∿o, edges in E_4.

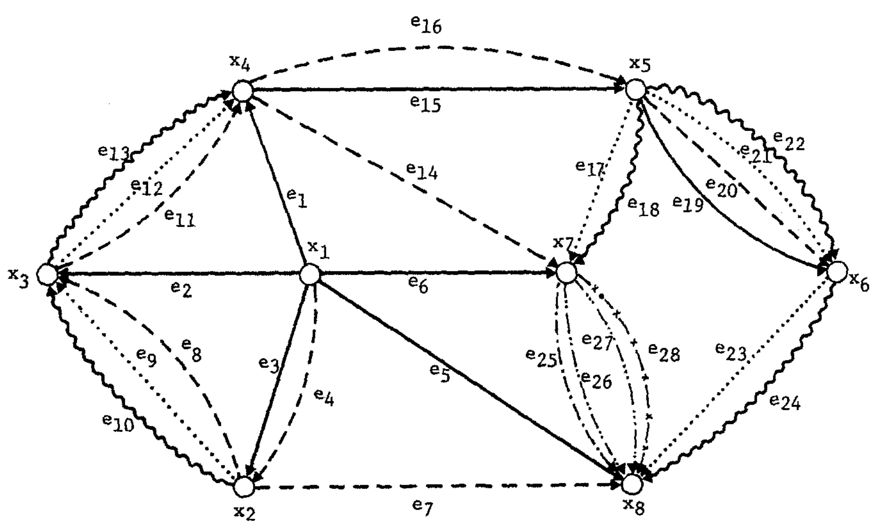

Fig. 3. Partition E_i of G^2 obtained by FOREST. o——o, E_1; o– – –o, E_2; o·········o, E_3; o∿∿∿o, E_4; o–·–·–o, E_5; o–··–··–o, E_6; o–···–···–o, E_7; o–x–x–o, E_8.

Note that the largest $r(x)$ is always chosen for the for-loop of line 6. In other words:

(2.2) When edge e is scanned, one of the end nodes of e has the largest label r in G.

FOREST without line 8 also works correctly, because once a node x is selected in line 5, all unscanned edges incident to x are scanned and x will never be referred to again. However, updating $r(x)$ in line 8 is useful for proving properties of FOREST as discussed below.

To find an unscanned node x with the largest $r(x)$ efficiently, we prepare $|V|$ buckets such that each unscanned node v is contained in the $r(v)$th bucket. All nonempty buckets are doubly linked by pointers so that an unscanned node x with the largest $r(x)$ can be found in $O(1)$ time and the link update after increasing label r of a node by one can also be done in $O(1)$ time. The entire time required to update bucket links is therefore $O(|V| + |E|)$ because labels are changed $O(|E|)$ times. This shows that time complexity of FOREST is $O(|V| + |E|)$.

We now show that each $F_i = (V, E_i)$, $1 \le i \le |E|$, computed by FOREST is a maximal spanning forest in $G - E_1 \cup E_2 \cup \cdots \cup E_{i-1}$.

LEMMA 2.2. *Consider the time instant when the begin–end block of lines* 7–10 *in FOREST has been completed for the current* x. *Let* $v \in V$ *be any node and let* $E(v)$ *denote the set of edges incident to* v. *Then*

$$E(v) \cap E_i \neq \varnothing \quad \textit{for} \quad i = 1, 2, \ldots, r(v),$$

$$E(v) \cap E_i = \varnothing \quad \textit{for} \quad i = r(v) + 1, \ldots, |E|.$$

PROOF. Immediate from FOREST. □

LEMMA 2.3. *All subgraphs* $F_i = (V, E_i)$, $i = 1, 2, \ldots, |E|$, *constructed by FOREST are forests.*

PROOF. At the instant of adding an edge $e = (x, y) \in E(y)$ to E_i in FOREST, $r(x) \ge r(y) = i - 1$ and hence $E(y) \cap E_i = \varnothing$ (by Lemma 2.2) holds. This proves that F_i does not contain a cycle, i.e., is a forest. □

It will now be shown that each forest F_i is maximal in $G - E_1 \cup E_2 \cup \cdots \cup E_{i-1}$. If an edge $e = (u, v)$ with $E(u) \cap E_i = \varnothing$ and $E(v) \cap E_i = \varnothing$ $(i \ge 1)$ is added to E_i at line 7 of FOREST, then such e is called a root edge of E_i. That is, by Lemma 2.2, the two end nodes u, v of such e satisfy $r(u) = r(v) = i - 1$ before e is added to E_i. During computation, each (V, E_i) may contain more than one nontrivial tree, where nontrivial means that the tree contains at least one edge. Each nontrivial tree T has exactly one root edge (i.e., the first edge of the tree), because FOREST adds to E_i no edge $e = (u, v)$ with $E(u) \cap E_i \neq \varnothing$ and $E(v) \cap E_i \neq \varnothing$ (i.e., T grows without merging with other trees). For example, in Figure 2 each of

(V, E_1), (V, E_2), and (V, E_4) has one nontrivial tree, (V, E_3) contains two nontrivial trees, and e_1, e_9, e_{15}, e_{19}, e_{29} are root edges.

If an edge $e = (u, v)$ is scanned in FOREST while scanning node u, we regard e as a directed arc $\vec{e} = u \to v$ (i.e., u is scanned before v). For example, in Figure 2 e_1 is illustrated as a directed arc from x_1 to x_2, since edge $e_1 = (x_1, x_2)$ is scanned while scanning node x_1. For a set of scanned edges $E' \subseteq E$ and $G' = (V, E')$, denote $\vec{E}' = \{\vec{e} \mid e \in E'\}$ and $\vec{G}' = (V, \vec{E}')$. $\vec{G}'$ is a directed graph. The directed tree $\vec{T}$ corresponding to a nontrivial tree T in (V, E_i) is always a rooted out-tree with root x such that $\vec{e} = x \to y$ is the root edge of T. This means that indeg(v) of T is at most one for any node v. Since all unscanned edges incident to x chosen at line 5 of FOREST become scanned, the resulting directed graph $\vec{G} = (V, \vec{E}_1 \cup \vec{E}_2 \cup \cdots \cup \vec{E}_{|E|})$ is acyclic. These properties are easily confirmed in Figure 2.

LEMMA 2.4.

(a) *When FOREST adds an edge $e = (u, v)$ to E_i at line 7, there exists a path $P_{i-1} \subseteq E_{i-1}$ connecting u and v.*

(b) *Let E_k $(k = 1, 2, \ldots, |E|)$ be the sets obtained by FOREST at some time instant. If there is a path $P_j \subseteq E_j$ connecting nodes u and v, then there are paths $P_i \subseteq E_i$ connecting the same u and v, for all $i < j$.*

PROOF. (a) By Lemma 2.2, $E(u) \cap E_{i-1} \neq \emptyset$ and $E(v) \cap E_{i-1} \neq \emptyset$. If there is no path $P_{i-1} \subseteq E_{i-1}$ connecting u and v, (V, E_{i-1}) has two nontrivial trees $\vec{T}_u$ and $\vec{T}_v$ containing u and v, respectively. Let u_0 (v_0) be the root in T_u (T_v), where we assume without loss of generality that u_0 has been scanned before v_0. Let $u_0, u_1, \ldots, u_h$ $(=u)$ denote the nodes in the unique path from u_0 to u in $\vec{T}_u$. After scanning u_0, u_1 has label $r \geq i - 1$, but v_0 has label $r \leq i - 2$ as discussed after Lemma 2.3, since v_0 is a root in E_{i-1}. Therefore u_1 is scanned before v_0. u_2 then has label $r \geq i - 1$. Repeating this argument, we see that $u_2, u_3, \ldots, u_h$ $(=u)$ and v are scanned before v_0. This contradicts the assumption that v_0 is the root of T_v (i.e., v_0 is scanned before v).

(b) Let e^k, $k = 1, 2, \ldots, h$, denote the edges in path $P_j \subseteq E_j$ $(j \geq 2)$, which connects u and v. By (a), each e_k has a path $P^k_{j-1} \subseteq E_{j-1}$ connecting the two end nodes of e^k. Clearly, $P^1_{j-1} \cup P^2_{j-1} \cup \cdots \cup P^h_{j-1}$ contains a path $P_{j-1} \subseteq E_{j-1}$ connecting u and v. Similarly for $i = j - 2, j - 3, \ldots, 1$. □

LEMMA 2.5. *Consider the sets E_i, $i = 1, 2, \ldots, |E|$, obtained by FOREST for a graph $G = (V, E)$. Then each (V, E_i) is a maximal spanning forest in $G - E_1 \cup E_2 \cup \cdots \cup E_{i-1}$.*

PROOF. Let $H_{i-1} = G - E_1 \cup E_2 \cup \cdots \cup E_{i-1}$. Every (V, E_i) is a forest by Lemma 2.3. If (V, E_i) is not maximal in H_{i-1}, there is an edge $e = (u, v) \in E_j$ for some $j > i$ such that $(V, E_i \cup \{e\})$ is a forest. However this contradicts Lemma 2.4(b). □

THEOREM 2.1. *Given a graph $G = (V, E)$, a partition E_i $(i = 1, 2, \ldots, |E|)$ of E satisfying* (2.1) *is found in $O(|V| + |E|)$ time, where $|E_i| \leq |V| - i$ $(i = 1, 2, \ldots,$

$|V|-1)$ *and* $|E_i| = 0$ $(i = |V|, \ldots, |E|)$ *if* G *is simple, and* $|E_i| \le |V| - 1$ $(i = 1, 2, \ldots, |E|)$ *if* G *is multiple.*

PROOF. By the discussion so far, it suffices to show that $|E_i| \le |V| - i$ $(i = 1, 2, \ldots, |V|-1)$ and $|E_i| = 0$ $(i = |V|, \ldots, |E|)$ if G is simple. Denote all nodes in G by $x_1, x_2, \ldots, x_{|V|}$ in the order scanned by FOREST. Since G has no multiple edges, $r(x_i)$ increases at most by 1 when an incident node x_k, $k < i$, is scanned. Hence, $r(x_i) \le i$, $i = 1, 2, \ldots, |V|$. This and Lemma 2.2 imply $E(x_i) \subseteq E_1 \cup E_2 \cup \cdots \cup E_i$. Therefore each (V, E_i) has at least $i - 1$ ($|V|$ if $i - 1 > |V|$) isolated nodes $x_1, x_2, \ldots, x_{i-1}$, implying $|E_i| \le |V| - i$ for $i \le |V| - 1$ and $|E_i| = 0$ for $i \ge |V|$. □

Consequently, a k-edge-connected spanning subgraph

$$G_k = (V, E' = E_1 \cup E_2 \cup \cdots \cup E_k)$$

of a k-edge-connected graph $G = (V, E)$ can be found in $O(|V| + |E|) = O(|E|)$ time, where $|E'| \le k|V| - k(k+1)/2$ if G is simple, or $|E'| \le k(|V| - 1)$ if G is multiple. These bounds on $|E'|$ are sharp since $|E| = |E'| = k(k+1) - k(k+1)/2 = k(k+1)/2$ if $G = (V, E) = K_{k+1}$ (i.e., a simple complete graph with $k+1$ nodes), and $|E| = |E'| = k(|V| - 1)$ if $G = (V, E)$ is k-multiple tree.

Note also that edge-connectivity of the above G_k is exactly k, provided that $\lambda(G) \ge k$, since G_k contains a node with degree $= k$ as we shall see below.

LEMMA 2.6. *For a graph* $G = (V, E)$, *let* E_i $(i = 1, 2, \ldots, |E|)$ *be obtained by FOREST. Then each* $G_k = (V, E_1 \cup E_2 \cup \cdots \cup E_k)$, $k \le \delta(G)$, *contains a node with degree* $= k$, *where* $\delta(G)$ *is the minimum degree of* G.

PROOF. Consider the last node x scanned by FOREST. Since all edges incident to x have been already scanned at the time of scanning x, there is no directed arc outgoing from x in $\vec{G}$. Since each (V, E_i) has no node v with $\text{indeg}(v) > 1$, each E_i with $i \le |E(x)|$ contains exactly one edge incident to x. Thus, $\deg(x) = k$ holds in G_k if $k \le |E(x)|$. □

3. k-Node-Connected Subgraph. In this section we consider node-connectivity. Let $\kappa(x, y; G)$ denote the local node-connectivity between nodes x and y in G, where $\kappa(x, y; G) = |V| - 1$ if x and y are adjacent. The node connectivity of G is defined by $\kappa(G) = \min\{\kappa(x, y; G) | x, y \in V\}$. Surprisingly, the spanning subgraph $G_k = (V, E_1 \cup E_2 \cup \cdots \cup E_k)$ obtained for a simple graph G by FOREST is k-node-connected for every $k \le \kappa(G)$. In the following, any graph is assumed to be simple, and an edge e with the end nodes $\{u, v\}$ is also denoted by an unordered pair (u, v).

Our goal is to prove the following theorem.

THEOREM 3.1. *For a given simple graph* $G = (V, E)$, *let* E_i $(i = 1, 2, \ldots, |E|)$ *be obtained by FOREST upon completion. Then each spanning subgraph*

$G_i = (V, E_1 \cup E_2 \cup \cdots \cup E_i)$ *satisfies*

$$\kappa(x, y; G_i) \geq \min\{\kappa(x, y; G), i\} \qquad \textit{for any} \quad x, y \in V. \tag{3.1}$$

Before proving Theorem 3.1, we need the following lemmas which are useful in understanding the dynamics of FOREST. In order to avoid confusion, we use notations E_i $(i = 1, 2, \ldots, |E|)$ to denote the final partition obtained from $G = (V, E)$ by FOREST, and E_i^* $(i = 1, 2, \ldots, |E|)$ to denote the intermediate edge sets E_i constructed by FOREST at a given time instant ($E_i^* = E_i$ holds for any i upon completion of FOREST).

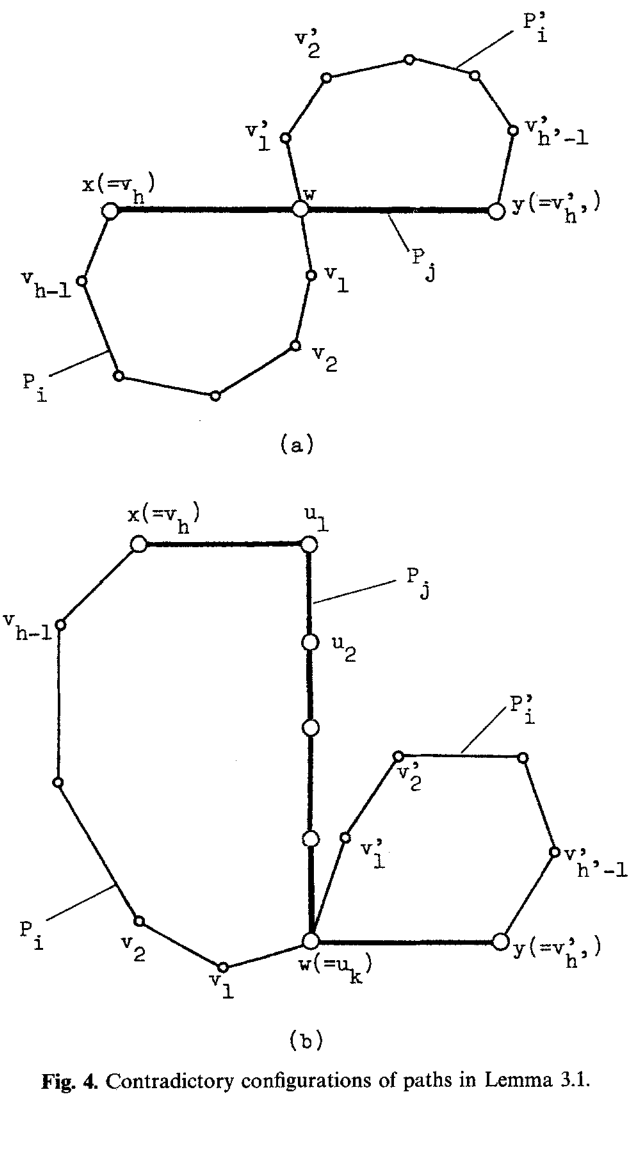

Fig. 4. Contradictory configurations of paths in Lemma 3.1.

LEMMA 3.1. *Consider a time instant during the execution of FOREST, and assume that there is an x–y path $P_j \subseteq E_j^*$ with the nodes*

$$x, u_1, u_2, \ldots, u_k = w, y \tag{3.2}$$

such that either $k = 1$ (Figure 4(a)) or u_1 is scanned before scanning $u_k = w$, if $k \geq 2$ (Figure 4(b)). If there are a w–x path $P_i \subseteq E_i^*$ and a w–y path $P_i' \subseteq E_i^*$, where $1 \leq i < j$, then these two paths contain a common node other than w.

PROOF. See the Appendix. □

LEMMA 3.2. *Consider a node cut set W in $G_{i+1} = (V, E_1 \cup E_2 \cup \cdots \cup E_{i+1})$, where the nodes in W are denoted $w_1, w_2, \ldots, w_i$ in the order scanned in FOREST (see Figure 5). Let X be a component in $G_{i+1} - W$, and let Y denote the rest of the components (note that $G_{i+1} - W$ may have more than two components). For each of $w_1, w_2, \ldots, w_i$, the following* (a) *and* (b) *hold immediately after FOREST has scanned $w_t \in W$ in FOREST, where E_j^* denotes the edge set constructed by FOREST at that time instant.*

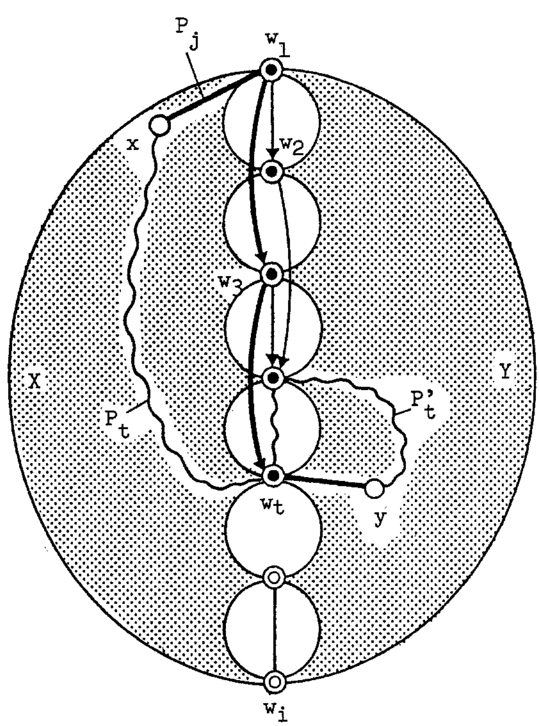

Fig. 5. Proof of Lemma 3.2. ⦿, scanned nodes in W; ◎, unscanned nodes in W.

(a) *Every path $P_t \subseteq E_t^*$ connecting a node in X to a node in Y passes through w_t.*
(b) *For any j such that $t + 1 \leq j \leq i + 1$, E_j^* has no path connecting a node in X to a node in Y.*

PROOF. Immediately after scanning $w_t \in W$, it is clear that

(3.3) if there is an edge $(w_h, w_{h'}) \in E$ with $h \leq t$ or $h' \leq t$, then $(w_h, w_{h'})$ has already been scanned and its direction is $w_h \to w_h$, $(w_{h'} \to w_h)$ if $h < h'$ $(h' < h)$.

However, at this instant,

(3.4) $\quad w_h \to p \quad$ with $\quad h \geq t + 1 \quad$ is not present yet for any $\quad p \in V$.

Now consider a path $P_j \subseteq E_j^*$ $(\subseteq E_j)$ such that it connects a node in X to a node in Y and $j \leq i + 1$. By definition of W, it can be assumed without loss of generality that all the nodes in P_j except fro x and y are all in W. Therefore denote the nodes in P_j from x to y by (see Figure 5)

(3.5) $$x, w_{i_1}, w_{i_2}, \ldots, w_{i_k}, y.$$

If $w_{i_g} \in \{w_{t+1}, w_{t+2}, \ldots, w_i\}$ for some g, then two edges (w_{i_g}, p_1) and (w_{i_g}, p_2) (with their orientation ignored) in P_j satisfy either $w_{i_g} \to p_1$ or $w_{i_g} \to p_2$ by indeg$(w_{i_g}) \leq 1$ in $\vec{P}_j$. This contradicts (3.4). Therefore,

(3.6) $$w_{i_g} \in \{w_1, w_2, \ldots, w_t\} \qquad \text{for} \quad g = 1, 2, \ldots, k.$$

This means that we have

$$t \geq i_1 > i_2 > \cdots > i_h < i_{h+1} < \cdots < i_k \leq t \qquad \text{for some} \quad 1 \leq h \leq k,$$

by properties (3.3) and (3.6). In the subsequent discussion we assume

(3.7) $$i_1 < i_k \qquad \text{if} \quad k \geq 2$$

without loss of generality. We now prove (a) and (b) by induction on t.

(I) Immediately after scanning $w_1 \in W$: Condition (a) for w_1 is obvious from (3.6) with $t = 1$. To prove (b), take a path $P_j \subseteq E_j^*$ as discussed above for j with $2 \leq j \leq i + 1$. By (3.7), the nodes in P_j are written as x, w_1, y. Also by Lemma 2.4(b), there exist an x–w_1 path $P_1 \subseteq E_1^*$ and a w_1–y path $P_1' \subseteq E_1^*$. From condition (a), whose validity with $t = 1$ was already shown, these P_1 and P_1' are node-disjoint except at w_1 (otherwise there is a path from X to Y in E_1^* not passing w_1). However, these P_j, P_1, and P_1' form a contradiction to Lemma 3.1, proving condition (b) for w_1.

(II) Immediately after scanning $w_t \in W$ $(t \geq 2)$: By induction hypothesis, condition (b) holds for w_{t-1}. That is, (3.8) at the time of having scanned w_{t-1}, E_j^* with $(t-1)+1 \leq j \leq i+1$ contains no path connecting a node in X to a node in Y.

Assume, after scanning w_t, that there is a path $P_j \subseteq E_j^*$ with $t \leq j \leq i+1$ that connects some $x \in X$ and $y \in Y$. Assume (3.5) and (3.7) for this P_j. By (3.8), we can assume $i_k = t$ (i.e., condition (a) for w_t is shown). Now assume that j further satisfies $t+1 \leq j$ $(\leq i+1)$ (i.e., this P_j does not satisfy (b)). By Lemma 2.4, there exist an x–w_t path $P_t \subseteq E_t^*$ and a w_t–y path $P_t' \subseteq E_t^*$. These P_t and P_t' are node-disjoint except at w_t by the above condition (a) for w_t. However, such P_j, P_t, P_t' form a contradiction to Lemma 3.1, and prove condition (b) for w_t. □

Proof of Theorem 3.1. We prove (3.1) by induction on i. Validity for $i = 1$ is obvious. Assuming that some $x, y \in V$ and i satisfy $\kappa(x, y; G_i) \geq \min\{\kappa(x, y; G), i\}$ and $\kappa(x, y; G_{i+1}) < \min\{\kappa(x, y; G), i+1\}$, we derive a contradiction. The assumption implies $\kappa(x, y; G) \geq i+1$ (then $i+1 \leq |V|-1$) and $\kappa(x, y; G_i) = \kappa(x, y; G_{i+1}) = i$. Note that x and y are not adjacent in G_{i+1} because $\kappa(x, y; G_{i+1}) = i < |V|-1$. $\kappa(x, y; G_{i+1}) = i$ implies that x and y are disconnected in $G_{i+1} - W$ for some node cut set $W \subseteq V - \{x, y\}$ satisfying $|W| = i$. Let X be the component containing x in $G_{i+1} - W$, and let Y denote the rest of components. This is illustrated in Figure 6. By $\kappa(x, y; G) \geq i+1$,

$$E'' = E_{i+2} \cup E_{i+3} \cup \cdots \cup E_{|E|}$$

contains an edge $e = (u, v) \in E_h$ for some $u \in X$, $v \in Y$, and $h \geq i+2$. Thus, by Lemma 2.4, there is a path $P_{i+1} \subseteq E_{i+1}$ connecting u and v. This P_{i+1} must pass

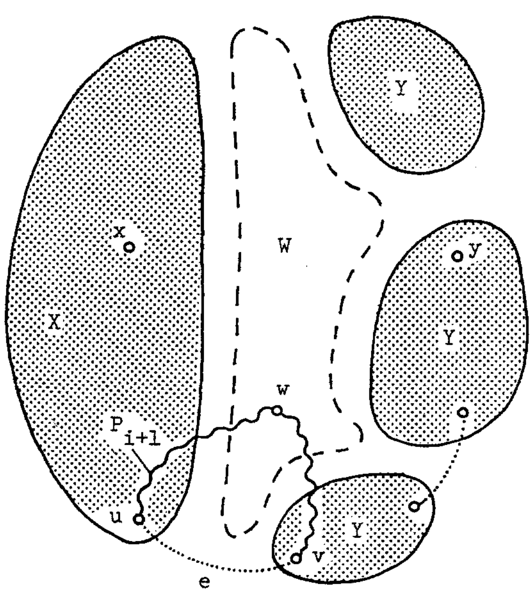

Fig. 6. Connected components in $G - E'' - W$. o·········o, edges in E''.

through a node w in W by definition of W. However, by Lemma 3.2(b), for w_i, we see that there is no such path $P_{i+1} \subseteq E^*_{i+1}$ after scanning all nodes in W. Hence there is no such $P_{i+1} \subseteq E_{i+1}$ since all edges incident to nodes in W have already been scanned at the time of scanning w_i. This is a contradiction and proves Theorem 3.1. □

4. Concluding Remarks. A linear-time algorithm for finding a sparse k-connected spanning subgraph for a given k-connected graph is developed in this paper. This is very useful in improving the time complexity of some algorithms for solving other graph problems, by preprocessing the given graph by FOREST. For example, an $O(\max\{k^2|V|^{1/2}, k|V|\}|E|)$-time algorithm [2] for testing whether $\kappa(G) \geq k$ can be improved to $O(\max\{k^2|V|^{1/2}, k|V|\}k|V|) = O(\max\{k^3|V|^{3/2}, k^2|V|^2\})$. This is an improvement since $O(k|V|) \leq O(|E|)$ can be assumed, as $k|V|/2 > |E|$ trivially implies $\kappa(G) < k$. To attain this, the spanning subgraph $G_k = (V, E_1 \cup E_2 \cup \cdots \cup E_k)$ is first computed by applying FOREST to $G = (V, E)$. This requires $O(|V| + |E|)$ time, and $G_k = (V, E')$ satisfies $\lambda(G_k) = \min\{\lambda(G), k\}$ and $|E'| = O(k|V|)$. Then $\kappa(G_k) \geq k$ ($\kappa(G) \geq k$) can be checked in $O(\max\{k^2|V|^{1/2}, k|V|\}|E'|) = O(\max\{k^3|V|^{3/2}, k^2|V|^2\})$ time by the algorithm in [2]. The total time is $O(|V| + |E| + \max\{k^3|V|^{3/2}, k^2|V|^2\}) = O(\max\{k^3|V|^{3/2}, k^2|V|^2\})$.

Based on this time complexity for testing $\kappa(G) \geq k$ and Matula's careful binary search [4], the current best bound $O(\max\{\kappa(G)^2|V|\,|E|, \kappa(G)|V|^{1/2}|E|\})$ to compute $\kappa(G)$ can be reduced to $O(\max\{\kappa(G)^3|V|^{3/2}, \kappa(G)^2|V|^2\})$. For this, we check $\kappa(G) \geq k$ for each $k = 2, 2^2, 2^3, \ldots$ in order to find the integer i satisfying $2^i \leq \kappa(G) < 2^{i+1}$. Since the algorithm [2] for testing $\kappa(G) \geq k$ provides $\kappa(G)$ if $\kappa(G) \leq k$, $\kappa(G)$ is obtained when such i is found. The total time is

$$\begin{aligned} &O(\max\{2^3|V|^{3/2}, 2^2|V|^2\}) + O(\max\{(2^2)^3|V|^{3/2}, (2^2)^2|V|^2\}) \\ &\quad + \cdots + O(\max\{(2^{i+1})^3|V|^{3/2}, (2^{i+1})^2|V|^2\}) \\ &\quad = O(\max\{|V|^{3/2}\cdot 8(8^{i+1} - 1)/(8 - 1), |V|^2\cdot 4(4^{i+1} - 1)/(4 - 1)\}) \\ &\quad = O(\max\{\kappa^3(G)|V|^{3/2}, \kappa^2(G)|V|^2\}). \end{aligned}$$

Based on the fact that the spanning subgraph $G_i = (V, E_1 \cup E_2 \cup \cdots \cup E_i)$ obtained by FOREST preserves the local edge- and node-connectivities up to i, we can improve the complexity for computing the number λ_{st} (κ_{st}) of edge (node) disjoint paths between specified nodes s and t in G. (Clearly, by Menger's theorem, $\lambda_{st} = \lambda(s, t; G)$ and

$$\kappa_{st} = \begin{cases} \kappa(s, t; G) & \text{if } s \text{ and } t \text{ are not adjacent,} \\ \kappa(s, t; G - \{(s, t)\}) + 1 & \text{if } s \text{ and } t \text{ are adjacent,} \end{cases}$$

where G is assumed to be simple.) For example, the algorithms in [1] determine whether $\lambda_{st} \geq k$ in $O(\min\{k, |V|^{2/3}\}|E|)$ time if G is simple, and in $O(\min\{|V|, k, |E|^{1/2}\}|E|)$ time if G is multiple, and $\kappa_{st} \geq k$ in $O(\min\{k, |V|^{1/2}\}|E|)$. By preprocessing G by FOREST, if $k(|V| - 1) \leq |E|$, these can be improved to $O(\min\{k^2|V|,$

$k|V|^{5/3}\}$), $O(\min\{k|V|^2, k^2|V|, k^{3/2}|V|^{3/2}\})$, and $O(\min\{k^2|V|, k|V|^{3/2}\})$, respectively. Based on these improved bounds and the above binary search, λ_{st} can be computed in $O(|E| + \min\{\lambda_{st}^2|V|, \lambda_{st}|V|^{5/3}\})$ time for a simple graph and in $O(|E| + \min\{\lambda_{st}|V|^2, \lambda_{st}^2|V|, \lambda_{st}^{3/2}|V|^{3/2}\})$ time for a multiple graph, and κ_{st} can be found in $O(|E| + \min\{\kappa_{st}^2|V|, \kappa_{st}|V|^{3/2}\})$ time.

Recently, further useful properties of FOREST have been studied [5, Theorem 2.2], [6], [7]. For example, [6] contains $O(|E| + \min\{\lambda(G)|V|^2, p|V| + |V|^2 \log|V|\})$ and $O(|V|\,|E| + |V|^2 \log|V|)$-time algorithms for determining the edge-connectivity of given multiple and capacitated graphs, respectively, where p ($\le |E|$) is the number of pairs of nodes between which the multiple graph has an edge. These algorithms may provide new insight into connectivity problems as they do not rely on max-flow algorithms, different from the algorithms previously known.

Acknowledgments. We wish to thank Professor Takao Nishizeki of Tohoku University for his valuable discussion as well as the information about references [8] and [9], and the anonymous referee for his helpful comments.

Appendix. Proof of Lemma 3.1. Assume that P_i and P'_i are node-disjoint except at w. Denote the nodes in P_i (P'_i) by $w, v_1, v_2, \ldots, v_h = x$ ($w, v'_1, v'_2, \ldots, v'_{h'} = y$). P_j is given by (3.2). First, the case of $k = 1$ is considered (Figure 4(a)). Since G has no multiple edge, $h \ge 2$ and $h' \ge 2$. Since indeg(w) ≤ 1 in $(V, \vec{E}_j^*)$, as discussed in Section 2, $w \to y$ is assumed without loss of generality. If $w \to v'_1$, then $w \to v'_1 \to v'_2 \to \cdots \to v'_{h'}(=y)$ holds, since indeg(v'_a) ≤ 1 in $\vec{E}_i^*$ for any v'_a. This implies that nodes $v'_1, v'_2, \ldots, v'_{h'-1}, y$ are all unscanned when w is scanned. Note that $r(y) < i$ holds when $(v'_{h'-1}, y)$ is added to E_i^* by scanning $v'_{h'-1}$. However, edge (w, y) has been scanned at the time of scanning w, and $(w, y) \in E_j^*$ means that $r(y) \ge j > i$ was satisfied after scanning w. This is a contradiction. Therefore, we assume $v'_1 \to w$. By a similar argument, $v'_1 \to w \to v_1 \to v_2 \to \cdots \to v_h(=x)$ follows from $v'_1 \to w$. Since $w \to x$ leads to a contradiction in the same manner as above, $x \to w$ is concluded. However, this creates a directed cycle in $\vec{P}_j$ and $\vec{P}_i$, again a contradiction to $\vec{G}_{|E|} = (V, \vec{E}_1 \cup E_2 \cup \cdots \cup \vec{E}_{|E|})$ is acyclic.

Now consider the case of $k \ge 2$ (Figure 4(b)). We have $w \to y$ by indeg(w) ≤ 1 in $(V, \vec{E}_j^*)$. Since $w \to v'_1$ is not possible for the same reason as the case of $k = 1$, we have $v'_1 \to w \to v_1 \to v_2 \to \cdots \to v_h(=x)$. From this, we see that w is scanned before v_{h-1}. To avoid a directed cycle, $u_1 \to x$ is concluded. Clearly, $(v_{h-1}, x) \in E_i^*$ implies that $r(x) < i$ when (v_{h-1}, x) is added to E_i^*, i.e., v_{h-1} is scanned before scanning u_1. This contradicts the fact that u_1 has been scanned before w. $\square$

References

[1] S. Even and R. E. Tarjan, Network flow and testing graph connectivity, *SIAM J. Comput.* **4** (1975), 507–518.

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

[3] M. R. Garey and D. S. Jhonson, *Computer and Intractability, A Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979.

[4] D. W. Matula, Determining edge connectivity in $O(nm)$, *Proceedings of the 28th Symposium on Foundations of Computer Science* (1987), pp. 249–251.

[5] H. Nagamochi and T. Ibaraki, Linear time algorithms for finding k-edge-connected and k-node-connected spanning subgraphs, Technical Report #89006, Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University (1989).

[6] H. Nagamochi and T. Ibaraki, Computing edge-connectivity in multiple and capacitated graphs, Technical Report #89009, Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University (1989).

[7] H. Nagamochi, Z. Sun, and T. Ibaraki, Counting the number of minimum cuts in multiple undirected graphs, Technical Report #89010, Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University (1989).

[8] T. Nishizeki and S. Poljak, Highly connected factors with a small number of edges, Working Paper (1989).

[9] H. Suzuki, N. Takahashi, and T. Nishizeki, An algorithm for finding a triconnected spanning subgraph, SIGAL Research Report 7-3 (in Japanese) (1989).