SIAM J. COMPUT.
Vol. 10, No. 2, May 1981

0094-5397/81/1002-0002 $01.00/0

AN ALGORITHM FOR FINDING K MINIMUM SPANNING TREES*

N. KATOH†, T. IBARAKI‡ AND H. MINE‡

Abstract. This paper presents an algorithm for finding K minimum spanning trees in an undirected graph. The required time is $O(Km + \min(n^2, m \log\log n))$ and the space is $O(K+m)$, where n is the number of vertices and m is the number of edges. The algorithm is based on three subroutines. The first two subroutines are used to obtain the second minimum spanning tree in $O(\min(n^2, m\alpha(m, n)))$ steps, where $\alpha(m, n)$ is Tarjan's inverse of Ackermann's function [12] which is very slowly growing. The third one obtains the kth minimum spanning tree in $O(m)$ steps when the jth minimum spanning trees for $j = 1, 2, \cdots, k-1$ are given.

Key words. kth minimum spanning tree, graph algorithm, computational complexity

1. Introduction. Let $G = (V, E)$ be an undirected connected graph with no parallel edges, where V is a set of n vertices and E is a set of m edges. The weight $w(e)$ is associated with each edge $e \in E$. The minimum spanning tree problem is to find the spanning tree T_1 with the minimum weight, where the weight of a spanning tree T (viewed as a set of edges) is defined by $w(T) = \sum_{e \in T} w(e)$. This problem has been intensively studied by many authors including Kruskal [9], Prim [11], Dijkstra [6], Yao [14], Cheriton and Tarjan [5]. Especially [5] and [14] require $O(m \log\log n)$ steps.

The jth minimum spanning tree T_j is defined recursively as follows.

(1) T_1 is the minimum spanning tree.

(2) T_j ($j \geqq 2$) is a spanning tree with the minimum weight among those different from $T_1, T_2, \cdots, T_{j-1}$.

Algorithms for finding K minimum spanning trees $T_1, T_2, \cdots, T_K$ have been studied by Burns and Haff [3], Camerini, Frata and Maffioli [4] and Gabow [7]. Gabow's algorithm requires $O(Km\alpha(m, n) + m \log n)$ steps and $O(K+m)$ space, where α is Tarjan's inverse of Ackermann's function [12] and is very slowly growing. We propose an algorithm with $O(Km + \min(m \log\log n, n^2))$ time and $O(K+m)$ space. This is slightly faster than Gabow's algorithm, and the required space is of the same order. The basic idea for enumeration is similar to but slightly different from Gabow's approach. This slight difference and the use of some additional lists make it possible to reduce the run time.

Section 2 gives the definition and a property of edge exchanges. Sections 3 and 4 give the outline and a detailed description of the entire algorithm. Section 5 analyzes time and space requirements. Sections 6 and 7 describe subroutines computing edge exchanges to generate the second and the jth minimum spanning trees, respectively. Although the algorithm explained in §§ 3 and 4 requires $O(Km)$ space, it is reduced to $O(K+m)$ in § 8.

2. T-exchanges. Let T be a spanning tree in G. A *T-exchange* is a pair of edges $[e, f]$ such that $e \in T$, $f \notin T$, and $T - e \cup f$ is a spanning tree. The *weight* of T-exchange $[e, f]$ is $w[e, f] = w(f) - w(e)$; note that the weight of tree $T - e \cup f$ is $w(T) + w[e, f]$.

LEMMA 2.1. [7] *A spanning tree T has minimum weight if and only if no T-exchanges have negative weight.*

* Received by the editors June 22, 1978, and in revised form April 11, 1980.

† Department of Medical Engineering, The Center for Adult Diseases, Osaka, Nakamichi Higashinariku, Osaka 537, Japan.

‡ Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University, Kyoto 606, Japan.

3. The outline of the algorithm. Our algorithm consists of routines GEN and GENK like Gabow's algorithm [7]. GEN computes T_j when $T_1, T_2, \cdots, T_{j-1}$ are given, using the branch-and-bound type technique described by Lawler [10]. Our GEN is different from Gabow's, however, in that a slightly different scheme is used to partition the solution space, and in that more information is stored in conjunction with solution space partition. GENK generates all the K minimum spanning trees using GEN as a subroutine.

The following lemma is a basis for our algorithm.

LEMMA 3.1. [7] *Let T be a minimum spanning tree satisfying the constraints $IN \subset T$ and $OUT \subset E - T$, where IN and OUT are given subsets of E. Then a minimum spanning tree which is different from T and satisfies the same constraint is given by $T - e \cup f$, where $[e, f]$ is a minimum T-exchange satisfying $e \in T - IN$ and $f \in E - T - OUT$.*

Now assume that the first $j-1$ $(j>1)$ minimum spanning trees have been generated. The set of remaining spanning trees is partitioned into $j-1$ disjoint sets

$$P_i^{j-1} = \{T_k | k > j-1, IN_i \subset T_k, OUT_i \subset E - T_k\}, \qquad i = 1, 2, \cdots, j-1,$$

where IN_i and $OUT_i (1 \leqq i < j)$ are set of edges which will be specified later (in a way slightly different from Gabow's definition). For $i = 1, 2, \cdots, j-1$, let

$$Q_i^{j-1} = \{([e, f], r) | \text{for each } f \in E - T_i - OUT_i, e \in T_i - IN_i$$
$$\text{gives the minimum } T_i\text{-exchange } [e, f] \text{ with weight } r = w[e, f]\}.$$

Note that each Q_i^{j-1} contains $|E - T_i - OUT_i| = O(m)$ labels therein.

Sets IN_i and OUT_i defining P_i^{j-1} $(1 \leqq i \leqq j-1)$ are given as follows. Initially, when $j = 2$ (i.e., only T_1 is obtained), IN_1 and OUT_1 defining P_1^{j-1} and Q_1^{j-1} are given by

$$IN_1 = \phi \quad \text{and} \quad OUT_1 = \phi.$$

In general, assume that T_j is obtained from T_{i^*} by applying T_{i^*}-exchange $[e^*, f^*]$. Then IN_i and OUT_i are updated as follows:

$$IN_{i^*} \leftarrow IN_{i^*}, \; OUT_{i^*} \leftarrow OUT_{i^*} \cup \{f^*\},$$
$$IN_j \leftarrow IN_{i^*} \cup \{f^*\}, \; OUT_j \leftarrow OUT_{i^*}.$$

Other IN_i and OUT_i do not change. These new sets define P_i^j for $i = 1, 2, \cdots, j$, and GEN computes the corresponding Q_i^j. Recall here that Gabow [7] uses the scheme $IN_{i^*} \leftarrow IN_{i^*} \cup \{e^*\}$, $OUT_{i^*} \leftarrow OUT_{i^*}$; $IN_j \leftarrow IN_{i^*}$, $OUT_j \leftarrow OUT_{i^*} \cup \{e^*\}$. Our definition is essential to make the subsequent computation possible.

By this definition, the next lemma is obvious.

LEMMA 3.2. *Let j be $2 \leqq j \leqq K$.*

(1) *For any $i = 1, 2, \cdots, j-1$, T_i is a minimum spanning tree satisfying $IN_i \subset T_i$ and $OUT_i \subset E - T_i$, and no other T_k $(k = 1, 2, \cdots, i-1, i+1, \cdots, j-1)$ satisfies this constraint.*

(2) *Any spanning tree T satisfies $IN_i \subset T$ and $OUT_i \subset E - T$ for exactly one i with $1 \leqq i \leqq j-1$.*

When $T_1, T_2, \cdots, T_{j-1}$ are known, Lemma 3.2(2) implies that T_j is given as a minimum spanning tree in $\bigcup_{i=1}^{j-1} P_i^{j-1}$ (note that P_i^{j-1} excludes $T_1, T_2, \cdots, T_{j-1}$). By Lemma 3.1 and Lemma 3.2(2), a minimum spanning tree in P_i^{j-1} is given by $T_i - e' \cup f'$, where $([e', f'], r')$ is a label in Q_i^{j-1} with the smallest r. Thus, if we let $([e^*, f^*], r^*)$ be a label in $\bigcup_{i=1}^{j-1} Q_i^{j-1}$ with the smallest $w(T_i) + r$, T_j is given by

$$T_j = T_{i^*} - e^* \cup f^*.$$

Using these i^*, e^* and f^*, new sets P_i^j and Q_i^j are defined and the computation proceeds as explained above.

Now we describe how to compute Q_i^j ($1 \leqq i \leqq j$) in GEN. While computing T_1, Q_1^1 is obtained in $O(\min(n^2, m\alpha(m, n)))$ steps by a special subroutine COMPQ1 or COMPQ2 depending upon whether $n^2 < m\alpha(m, n)$ or not. These subroutines are explained in § 6. When $T_j = T_{i^*} - e^* \cup f^*$ is obtained in GEN, $Q_{i^*}^j$ is computed by

$$Q_{i^*}^j = Q_{i^*}^{j-1} - \{([e^*, f^*], r^*)\}.$$

Q_i^j ($i \neq i^*, j$) are simply obtained by $Q_i^j = Q_i^{j-1}$. Finally Q_j^j is obtained by calling subroutine COMPQ3 explained in § 7. Obviously $|Q_i^j| = O(m)$ for all i. The key point is that $Q_{i^*}^j$ and Q_j^j are both obtained in $O(m)$ steps from $Q_{i^*}^{j-1}$. Based on these Q_i^j, minimum trees in P_i^j can also be obtained in $O(m)$ steps (note that only $P_{i^*}^j$ and P_j^j are considered since minimum spanning trees in P_i^{j-1} and P_i^j do not change for $i \neq i^*, j$). GEN is repeated for $j = 2, 3, \cdots, K$, and the entire procedure is organized as GENK.

Finally, we briefly explain the actual data structures of some of the above mentioned lists. Each set P_i^{j-1} is represented in our algorithm by a tuple

$$P_i^{j-1} = (t', [e', f'], A_i, IN_i, OUT_i, i),$$

where $([e', f'], r')$ is a label in Q_i^{j-1} with the smallest r, and $t' = w(T_i) + r' (= w(T_i - e' \cup f'))$. A_i is the adjacency list of T_i: $A_i = \{A_i(u) | u \in V\}$ and $A_i(u) = \{v | \text{edge } (u, v) \in T_i\}$. The length of P_i^{j-1} is $O(m)$ since $|A_i| = O(n)$ and $|IN_i| + |OUT_i| = O(m)$. In our implementation, more information is associated with $A_i(u)$; actually it consists of the following tuples:

$$A_i(u) = \{(v, w(u, v), \text{INFLAG}) | (u, v) \in T_i\},$$

where INFLAG $= 0$ implies $(u, v) \notin IN_i$ and INFLAG $= 1$ implies $(u, v) \in IN_i$. We also prepare an adjacency list of G: $A_G = \{A_G(u) | u \in V\}$, $A_G(u) = \{(v, w(u, v)) | (u, v) \in E\}$.

4. Algorithm for finding K minimum spanning trees. This section describes algorithms GENK and GEN in an ALGOL-like language.

Procedure *GENK*$(G = (V, E), K)$; **begin**
comment $K \geqq 2$;
Find the adjacency list A_1 for a minimum weight spanning tree T_1 and its weight t_1; **output** (A_1);
if $n^2 < m\alpha(m, n)$ **then** call *COMPQ*1 to obtain Q_1^1
else call *COMPQ*2 to obtain Q_1^1;
Find a minimum weight exchange $([e', f'], r')$ in Q_1^1;
Let P_1^1 be $(t_1 + r', [e', f'], A_1, \phi, \phi, 1)$;
For $j = 2$ **until** K **do** call *GEN*$(P_i^{j-1}, Q_i^{j-1} | i = 1, 2, \cdots, j-1)$;
end *GENK*;

Procedure *GEN*$(P_i^{j-1}, Q_i^{j-1} | i = 1, 2, \cdots, j-1)$; **begin**
Find $P_{i^*}^{j-1} = (t^*, [e^*, f^*], A_{i^*}, IN_{i^*}, OUT_{i^*}, i^*)$ with the smallest weight t' among $P_i^{j-1} = (t', [e', f'], A_i, IN_i, OUT_i, i)$, $i = 1, 2, \cdots, j-1$;
if $t^* = \infty$ **then stop** (all spanning trees have been output and G has only $j-1$ spanning trees);
else begin
$A_j \leftarrow A_{i^*}$ with edge e^* replaced by f^* (A_j is the adjacency list of T_j); **output** (A_j);
$Q_{i^*}^j \leftarrow Q_{i^*}^{j-1} - \{([e^*, f^*], t^* - t_{i^*})\}$;

5 Call $COMPQ3(A_j, IN_{i^*} \cup f^*, OUT_{i^*}, f^*, Q_{i^*}^{j-1})$ to obtain Q_j^j;
6 $Q_i^j \leftarrow Q_i^{j-1}$ for $i \neq i^*, j$;
7 **if** $Q_{i^*}^j \neq \phi$ **then** $P_{i^*}^j \leftarrow (t_{i^*}+r', [e', f'], A_{i^*}, IN_{i^*}, OUT_{i^*} \cup f^*, i^*)$, where $([e', f'], r')$ is a label in $Q_{i^*}^j$ with the minimum r and t_{i^*} (the weight of T_{i^*}) can be computed by $t_{i^*} = t^* - w[e^*, f^*]$
else $P_{i^*}^j \leftarrow (\infty, \phi, \phi, \phi, \phi, i^*)$;
8 **if** $Q_j^j \neq \phi$ **then** $P_j^j \leftarrow (t^*+r'', [e'', f''], A_j, IN_{i^*} \cup f^*, OUT_{i^*}, j)$, where $([e'', f''], r'')$ is a label in Q_j^j with the minimum r
else $P_j^j \leftarrow (\infty, \phi, \phi, \phi, \phi, i^*)$;
9 $P_i^j \leftarrow P_i^{j-1}$ for $i \neq i^*, j$;
end
return
end *GEN*

5. The correctness and the time bound of the algorithm.

LEMMA 5.1. *For each $j = 2, 3, \cdots, K$,* GEN *correctly computes T_j, Q_i^j and P_i^j $(i = 1, 2, \cdots, j)$ in $O(m)$ steps.*

Proof. Since the correctness follows from the result of [7] and the discussion given so far, we consider the time requirement only. Line 1 of GEN finds $P_{i^*}^{j-1}$ with the minimum t among P_i^{j-1}, $i = 1, 2, \cdots, j-1$. This is done in $O(\log(j-1)) \leqq O(\log K) = O(m)$ steps (since the number of spanning trees $\leqq 2^m$) if an appropriate sorting technique is used (e.g., heap sort [8]) for the set $\{P_i^{j-1} | i = 1, 2, \cdots, j-1\}$. Line 2 requires constant steps. Line 3 is done in $O(n)$ $(\leqq O(m))$ steps by $|A_{i^*}| = O(n)$. Line 4 requires $O(m)$ steps since $|Q_{i^*}^{j-1}| = O(m)$. Line 5 calls COMPQ3 and it requires $O(m)$, steps as will be shown in § 7. Lines 6 and 9 require constant steps because these are accomplished simply by keeping the previous data. Lines 7 and 8 are done in $O(m)$ steps by $|Q_{i^*}^j| = O(m)$ and $|Q_j^j| = O(m)$. (Adjustment of data structure of $\{P_i^j | i = 1, 2, \cdots, j\}$ (e.g., using heap) is also done in $O(\log j) \leqq O(m)$ steps, as is well known.) Thus, all computation in GEN is done in $O(m)$ steps. □

THEOREM 5.2. GENK *correctly generates the K minimum spanning trees from T_1 to T_K in $O(Km + \min(n^2, m \log\log n))$ time.*

Proof. The correctness of GENK follows from the previous discussion. The time requirement is analyzed here. Line 1 requires $O(\min(n^2, m \log\log n))$ steps (e.g., [5], [14]). Line 2 requires $O(\min(n^2, m\alpha(m, n)))$ steps as shown in § 6. Line 3 requires $O(m)$ steps by $|Q_1^1| = O(m)$. Line 4 requires constant time. Line 5 calls GEN $K-1$ times, and requires $O(Km)$ steps in total by Lemma 5.1. Thus, the total time is as shown above. □

A straightforward implementation of GENK requires $O(Km)$ space mainly to store Q_i^{j-1} and P_i^{j-1} for $i = 1, 2, \cdots, j-1$. This will be reduced to $O(K+m)$ in § 8.

6. Subroutines COMPQ1 and COMPQ2. This section briefly explains the two subroutines COMPQ1 and COMPQ2, computing Q_1^1 in $O(n^2)$ steps and in $O(m\alpha(m, n))$ steps respectively. These are based on the next lemma.

LEMMA 6.1. *For a given edge $f \in E - T_1$, let e be the maximum weight edge $(\neq f)$ on the unique cycle formed by adding f to T_1. Then $[e, f]$ is the smallest T_1-exchange with the given $f \in E - T_1$.*

Proof. The proof immediately follows since the weight of edge exchange $w[e, f]$ is given by $w(f) - w(e)$. □

Q_1^1 is therefore computed by finding $[e, f]$ and $r = w[e, f]$ of Lemma 6.1 for every edge $f = (u, v) \in E - T_1$. COMPQ1 is first outlined. It is a slight augmentation of Prim's algorithm [11] which computes T_1 in $O(n^2)$ time; we assume the reader's familiarity with Prim's algorithm. Consider a computation stage when an edge (x, v) is added to the current fragment (subtree which is going to comprise T_1), where x is a vertex in the fragment and v is not in the fragment. For each vertex u in the fragment, let (h, k) be the maximum weight edge in $u \overset{*}{\underset{T_1}{\rightarrow}} x$ (which has already been computed and stored). Then a maximum weight edge in $u \overset{*}{\underset{T_1}{\rightarrow}} v$ for $f = (u, v)$ is obtained by taking the edge with the larger weight between (x, v) and (h, k). (This property easily follows from Lemma 6.1 and is omitted.) Thus computation of such maximum weight edges for all (u, v), such that u's are vertices in the fragment is done in $O(n)$ time. Since this can be repeated $n-1$ times until T_1 is constructed by Prim's algorithm, computing at the same time the maximum weight edge for every $f = (u, v) \in E$, the total time to compute Q_1^1 is $O(n^2)$.

THEOREM 6.1. COMPQ1 *computes* Q_1^1 *in* $O(n^2)$ *time.* □

COMPQ2 is a straightforward adaptation of Tarjan's algorithm [13] for verifying in $O(m\alpha(m, n))$ time that a tree T in an undirected graph is a minimum spanning tree. His algorithm involves the computation of the maximum weight edge along the path $u \overset{*}{\underset{T}{\rightarrow}} v$, for each edge $f = (u, v) \notin T$. By Lemma 6.1, this portion of his algorithm can be directly used as COMPQ2 for computing Q_1^1.

THEOREM 6.2. COMPQ2 *computes* Q_1^1 *in* $O(m\alpha(m, n))$ *time.* □

7. Subroutine COMPQ3. This section describes subroutine COMPQ3(A_j, IN_j, OUT_j, f^*, $Q_{i^*}^{j-1}$) for obtaining Q_j^j in $O(m)$ steps when $T_j = T_{i^*} - e^* \cup f^*$ is given, where $[e^*, f^*]$ is the minimum T_{i^*}-exchange in $\bigcup_{i=1}^{j-1} Q_i^{j-1}$ (see § 3). COMPQ3 is based on the following lemma.

LEMMA 7.1. *For T_j and the edge $f^* = (u^*, v^*) \in T_j$ defined above, let $T_j(u^*)$ and $T_j(v^*)$ be two trees obtained from T_j by deleting f^*, where $u^* \in V_j(u^*)$ and $v^* \in V_j(v^*)$. Here $V_j(x)$ is the set of vertices in the connected component $T_j(x)$. Let $f = (u, v)$ be an edge in $E - T_j - OUT_j$.*

(1) *If $u, v \in V_j(u^*)$ or $u, v \in V_j(v^*)$, the label $([e, f], r)$ stored in $Q_{i^*}^{j-1}$ is also in Q_j^j.*

(2) *If $u \in V_j(u^*)$ and $v \in V_j(v^*)$, then $([e, f], r)$ stored in Q_j^j is determined by*

$$w(e) = \max\left[\max\left\{w(g) \mid g \notin IN_j,\ g \text{ is on } u^* \xrightarrow[T_j(u^*)]{*} u\right\},\right.$$

$$\left.\max\left\{w(h) \mid h \notin IN_j,\ h \text{ is on } v^* \xrightarrow[T_j(v^*)]{*} v\right\}\right],$$

$$r = w(f) - w(e).$$

Proof.

(1) Assume $u, v \in T_j(u^*)$ without loss of generality (see Fig. 1). Since $T_j - f^* = T_{i^*} - e^*$ (i.e., $T_j(u^*) = T_{i^*}(u^*)$), then $u \overset{*}{\underset{T_j}{\rightarrow}} v = u \overset{*}{\underset{T_{i^*}}{\rightarrow}} v$, and f^* is not an edge on $u \overset{*}{\underset{T_j}{\rightarrow}} v$. Thus, by Lemma 6.1 the label $([e, f], r)$ stored in $Q_{i^*}^{j-1}$ is also in Q_j^j.

(2) Since $u \overset{*}{\underset{T_j}{\rightarrow}} v$ is equal to $u \xrightarrow[T_j(u^*)]{*} u^* \rightarrow v^* \xrightarrow[T_j(v^*)]{*} v$ and $(u^*, v^*)(=f^*) \in IN_j$ (see Fig. 2), the edge e defining $([e, f], r) \in Q_j^j$ is the maximum weight edge $\notin IN_j$ on either $u \xrightarrow[T_j(u^*)]{*} u^*$ or $v^* \xrightarrow[T_j(v^*)]{*} v$ by Lemma 6.1. □

To compute $([e, f], r) \in Q_j^j$ efficiently by Lemma 7.1, COMPQ3 preprocesses trees $T_j(u^*)$ and $T_j(v^*)$ by calling subroutines EDGEFIND(A_j', u^*, IN_j) and EDGEFIND(A_j', v^*, IN_j), where A_j' is the adjacency list of $T_j(u^*) \cup T_j(v^*)$.

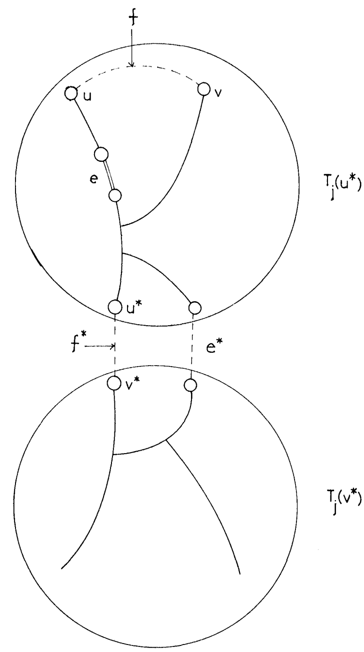

FIG. 1. *Illustration of T_j-exchange in the proof of case* (1) *of Lemma* 7.1.

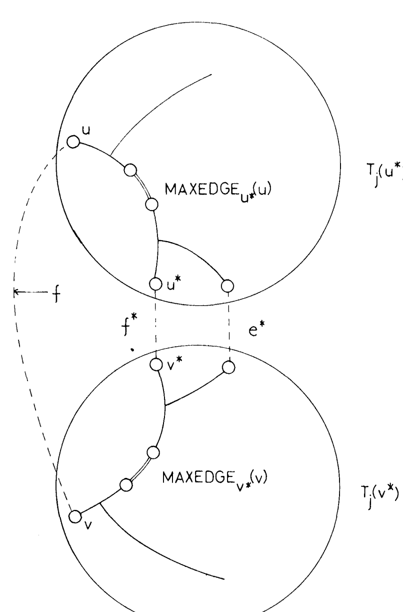

FIG. 2. *Illustration of T_j-exchange in the proof of case* (2) *of Lemma* 7.1.

EDGEFIND(A'_j, u^*, IN_j) finds the maximum weight edge $\text{MAXEDGE}_{u^*}(u) \in T_j(u^*) - IN_j$ on $u^* \xrightarrow[T_j(u^*)]{*} u$ for each vertex $u \in T_j(u^*)$. Its weight is stored in $W_{u^*}(u)$. (Subscript u^* is added to indicate MAXEDGE and W obtained by EDGEFIND(A'_j, u^*, IN_j).) EDGEFIND(A'_j, v^*, IN_j) is similar. After this, $([\text{MAXEDGE}_{w^*}(z), f], w(f) - W_{w^*}(z))$ is added to Q^j_j for each edge $f = (u, v) \in E - T_j - OUT_j$ with $u \in V_j(u^*)$ and $v \in V_j(v^*)$, where $W_{w^*}(z) = \max\{W_{u^*}(u), W_{v^*}(v)\}$.

For each $f = (u, v) \in E - T_j - OUT_j$ with $u, v \in V_j(u^*)$ or $u, v \in V_j(v^*)$, $([e, f], r)$ in $Q^{j-1}_{i^*}$ is directly stored in Q^j_j.

Procedure $COMPQ3(A_j, IN_j, OUT_j, f^* = (u^*, v^*), Q^{j-1}_{i^*})$; **begin**
1 $Q^j_j \leftarrow \phi$; $A'_j(u^*) \leftarrow A_j(u^*) - \{v^*\}$; $A'_j(v^*) \leftarrow A_j(v^*) - \{u^*\}$; $A'_j(u) \leftarrow A_j(u)$ for $u \neq u^*$, v^* ($A'_j = \{A'_j(u) \mid u \in V\}$ is the adjacency list of $T_j(u^*) \cup T_j(v^*)$);
2 **for** $u \in V$ **do**
3 **if** $u \in V_j(u^*)$ **then** $N(u) \leftarrow 1$ **else** $N(u) \leftarrow 0$ ($N(u)$ is a flag showing whether $u \in V_j(u^*)$ or $u \in V_j(v^*)$);
4 **for** $u \in V$ **do** $W_{u^*}(u) \leftarrow W_{v^*}(u) \leftarrow -\infty$; $MAXEDGE_{u^*}(u) \leftarrow MAXEDGE_{v^*}(u) \leftarrow \phi$;
5 Call $EDGEFIND(A'_j, u^*, IN_j)$ to obtain $MAXEDGE_{u^*}(u)$ and $W_{u^*}(u)$ for $u \in V_j(u^*)$;
6 Call $EDGEFIND(A'_j, v^*, IN_j)$ to obtain $MAXEDGE_{v^*}(v)$ and $W_{v^*}(v)$ for $v \in V_j(v^*)$;
7 **for** $f = (u, v) \in E - T_j - OUT_j$ **do**
8 **if** $N(u) = N(v)$ **then** add label $([e, f], r)$ in $Q^{j-1}_{i^*}$ to Q^j_j
9 **else** add label $([MAXEDGE_{w^*}(z), f], w(f) - W_{w^*}(z))$ to Q^j_j, where $z \in \{u, v\}$ satisfies $W_{w^*}(z) = \max(W_{u^*}(u), W_{v^*}(v))$ (if $W_{w^*}(z) = \infty$, do not add the label since $MAXEDGE_{w^*}(z)$ does not exist);
return
end $COMPQ3$;

Procedure $EDGEFIND(A'_j, p^*, IN)$; **begin**
Call $DFS(A'_j, \phi, p^*, IN)$;
return
end $EDGEFIND$;

Procedure $DFS(A'_j, x, y, IN)$; **begin**
for $z \in A'_j(y) - x (= A'_j(y)$ if $x = \phi)$ and $(y, z) \notin IN$ **do**
if $W(y) < w(y, z)$ **then begin**
$MAXEDGE(z) \leftarrow (y, z)$; $W(z) \leftarrow w(y, z)$;
end
else begin
$MAXEDGE(z) \leftarrow MAXEDGE(y)$; $W(z) \leftarrow W(y)$;
end
Call $DFS(A'_j, y, z, IN)$;
return
end DFS

THEOREM 7.2. COMPQ3(A_j, IN_j, OUT_j, f^*, $Q^{j-1}_{i^*}$) *correctly computes* Q^j_j *in* $O(m)$ *steps.*

Proof. It is obvious that $A'_j = \{A'_j(u) \mid u \in V\}$ obtained from A_j at line 1, is the adjacency list of $T_j(u^*) \cup T_j(v^*)$. Thus, EDGEFIND(A'_j, u^*, IN_j) and EDGEFIND(A'_j, v^*, IN_j) correctly compute $\text{MAXEDGE}_{u^*}(u)$, $W_{u^*}(u)$ for all $u \in V_j(u^*)$ and $\text{MAXEDGE}_{v^*}(v)$, $W_{v^*}(v)$ for all $v \in V_j(v^*)$. Thus, lines 7–9 correctly compute Q^j_j by Lemma 7.1. Next, we analyze the time requirement. Line 1 requires

$O(n)$ steps since $|A_j| = O(n)$. Lines 2 and 3 are done in $O(n)$ steps by computing $V_j(u^*)$ and $V_j(v^*)$ using A_j, and associating flag $N(u)$ to $u \in V$ by using A'_j. Line 4 is also done in $O(n)$ steps. Lines 5 and 6 require $O(n)$ steps, by $|V_j(u^*)| = O(n)$, $|V_j(v^*)| = O(n)$ and $|A_j| = O(n)$. To execute lines 7–9 in constant time for each $f = (u, v) \in E - T_j - OUT_j$, note that

$$E - T_j - OUT_j = \{f | ([e, f], r) \in Q_{i^*}^{j-1}\} - f^* \cup e^*$$

holds. Furthermore, $N(u) \neq N(v)$ holds for $e^* = (u, v)$. Thus, adding label $([e, f], r)$ in $Q_{i^*}^{j-1}$ to Q_j^j of line 8 is done in constant time if $f(\neq e^*)$ of line 7 is directly taken from $Q_{i^*}^{j-1}$ (i.e., constant time is required to search $([e, f], r)$ in $Q_{i^*}^{j-1}$ of line 8). Line 8, obviously, requires constant time. Thus, lines 7–9 require $O(m)$ steps in total by $|E| = m$. □

8. Space reduction. The required space for GENK is reduced to $O(K + m)$ in this section, although GENK, explained in § 4, requires $O(Km)$ space to store P_i^{j-1} and Q_i^{j-1} $(i = 1, 2, \cdots, j-1)$ (space required for other data is obviously $O(m)$).

First, in order to reduce the space requirement of P_i^{j-1} from $O(Km)$ to $O(K + m)$ we modify the data structure representing P_i^{j-1} in almost the same way as done by Gabow [7]. Namely, the data structure of P_1^{j-1} is the same as the one discussed in § 3 (P_1^{j-1} requires $O(m)$ space). P_i^{j-1} $(i > 1)$ are modified to

$$P_i^{j-1} = (t', [e', f'], [e^*(i), f^*(i)], i^*(i), b(i), s(i), i), \qquad i = 2, 3, \cdots, j-1$$

(thus, P_i^{j-1} $(i = 2, 3, \cdots, j-1)$ require $O(j) \leqq O(K)$ space), where t', $[e', f']$ are defined in § 3, and T_i is obtained from $T_{i^*(i)}$ by $T_{i^*(i)}$-exchange $[e^*(i), f^*(i)]$. The derivation of $T_2, T_3, \cdots$ from T_1 is represented by a rooted tree; T_{i^*} is a father of T_i (or T_i is a son of T_{i^*}) if T_i is derived from T_{i^*} by a T_{i^*}-exchange, and T_i and T_k are brothers if their fathers coincide. T_i is placed to the left of its brother T_k if $i < k$. $b(i)$ and $s(i)$ in P_i^{j-1} denote the brother $T_{b(i)}$ immediately to the left of T_i ($b(i) = 0$ if T_i is the leftmost brother) and the rightmost son $T_{s(i)}$. Obviously, T_1 is the root of this tree. Based on the new lists, T_i, IN_i and OUT_i can be constructed in $O(m)$ time by following the path from T_i up to root T_1. This technique is almost the same as Gabow's, and hence the details are omitted. The rest of the computation is then applied to the reconstructed P_i^{j-1}.

In order to reduce the space requirement of Q_i^{j-1} $(i = 1, 2, \cdots, j-1)$ to $O(K)$ we execute the following cleanup step from time to time. Note that each T_i-exchange $([e, f], r) \in Q_i^{j-1}$ induces a spanning tree $T_i - e \cup f$ with weight $w(T_i) + r$. However, only $K - (j-1)$ smallest spanning trees induced from $\bigcup_{i=1}^{j-1} Q_i^{j-1}$ are necessary to compute $T_j, T_{j+1}, \cdots, T_K$, as justified below. Therefore, the cleanup step removes all $([e, f], r)$'s from $\bigcup_{i=1}^{j-1} Q_i^{j-1}$, except those with $K - (j-1)$ smallest $w(T_i) + r$'s. The cleanup step is done in $O(K')$ steps, where $K' = |\bigcup_{i=1}^{j-1} Q_i^{j-1}|$, by finding the $K - (j-1)$th smallest element in $O(K')$ steps by the fast algorithm [2], and then removing all $([e, f], r)$'s with larger $(w(T_i) + r)$'s.

The cleanup step is justified as follows. Suppose that a T_i-exchange $[e, f]$ corresponding to $f \in E - T_i - OUT_i$ is removed from $Q_i^{i'-1}$ by a cleanup step. Later, a T_j may be obtained from T_i by $T_i - e^* \cup f^*$, and Q_j^j is computed from Q_i^{j-1} by COMPQ3. Note that Q_i^{j-1} is obtained from $Q_i^{i'-1}$ and therefore Q_j^j does not contain the minimum T_j-exchange $[e', f]$ corresponding to the removed f. At this point, it is necessary to show that such $[e', f]$ in Q_j^j can be ignored for the rest of the computation. However, this is obvious because $w(T_i - e \cup f) \leqq w(T_j - e' \cup f)$ can be easily proved and hence $T_j - e' \cup f$ is not a member of the K minimum spanning trees.

Now execute the cleanup step whenever $K' > 2K$ is satisfied. Then $K' = O(K)$ always holds. Since K' increases at most by m at every iteration, the cleanup step is necessary at every $\lceil K/m \rceil$ iterations, where $\lceil x \rceil$ denotes the smallest integer not smaller than x. Hence, the cleanup step is executed $O(K) \cdot O(m/K) = O(m)$ times before the entire computation is completed. Therefore, $O(Km)$ steps are required for the cleanup computation, but the time bound of the entire algorithm does not change.

Consequently, we have the next theorem.

THEOREM 8.1. GENK *requires* $O(Km + \min(n^2, m \log\log n))$ *time and* $O(K + m)$ *space.*

Acknowledgment. The authors would like to thank Professor H. N. Gabow of Colorado University for his helpful comments, especially for pointing out that Q_1^1 can be obtained in $O(m\alpha(m, n))$ time (in addition to mentioning Tarjan's result [13], he also proposed a new way of accomplishing this time bound, which is not included here).

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA., 1974.

[2] M. BLUM ET AL., *Time bounds for selection*, J. Comput. System Sci., 7 (1973), pp. 448–461.

[3] R. N. BURNS AND C. E. HAFF, *A ranking problem in graphs*, Proceedings of the 5th Southeast Conference on Combinatorics, Graph Theory and Computing, 19 (1974), pp. 461–470.

[4] P. M. CAMERINI, L. FRATTA AND F. MAFFIOLI, *The K shortest spanning trees of a graph*, Int. Rep. 73-10, IEEE-LCE Politechnico di Milano, Italy, 1974.

[5] D. CHERITON AND R. E. TARJAN, *Finding minimum spanning trees*, this Journal, 5 (1976), pp. 724–742.

[6] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[7] H. N. GABOW, *Two algorithms for generating weighted spanning trees in order*, this Journal, 6 (1977), pp. 139–150.

[8] D. E. KNUTH, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA., 1973.

[9] J. KRUSKAL, *On the shortest spanning subtree of a graph and the travelling salesman problem*, Proc. Amer. Math. Soc., 2 (1956), pp. 48–50.

[10] E. L. LAWLER, *A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem*, Management Sci., 18 (1972), pp. 401–405.

[11] R. C. PRIM, *Shortest connection networks and some generalizations*, Bell System Tech. J., 36 (1957), pp. 1389–1401.

[12] R. E. TARJAN, *Efficiency of a good but not linear set union algorithm*, J. Assoc. Comp. Mach., 22 (1975), pp. 215–225.

[13] ———, *Applications of path compression on balanced trees*, J. Assoc. Comp. Mach., 26 (1979), pp. 690–715.

[14] A. C. YAO, *An $O(|E| \log\log |V|)$ algorithm for finding minimum spanning trees*, Inform. Process. Lett., 4 (1975), pp. 21–23.