International Journal of Computer Mathematics
Vol. 87, No. 14, November 2010, 3175–3185

Listing all the minimum spanning trees in an undirected graph

Takeo Yamada*, Seiji Kataoka and Kohtaro Watanabe

Department of Computer Science, National Defense Academy, Yokosuka, Kanagawa 239-8686, Japan

(*Received 05 December 2008; revised version received 06 April 2009; second revision received 15 July 2009; accepted 06 September 2009*)

Efficient polynomial time algorithms are well known for the minimum spanning tree problem. However, given an undirected graph with integer edge weights, minimum spanning trees may not be unique. In this article, we present an algorithm that lists all the minimum spanning trees included in the graph. The computational complexity of the algorithm is $\mathrm{O}(N(mn + n^2 \log n))$ in time and $\mathrm{O}(m)$ in space, where n, m and N stand for the number of nodes, edges and minimum spanning trees, respectively. Next, we explore some properties of cut-sets, and based on these we construct an improved algorithm, which runs in $\mathrm{O}(Nm \log n)$ time and $\mathrm{O}(m)$ space. These algorithms are implemented in C language, and some numerical experiments are conducted for planar as well as complete graphs with random edge weights.

Keywords: minimum spanning tree; enumeration algorithm

2000 AMS Classifications: 05C05; 05C30; 68W05

1. Introduction

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{e_1, \ldots, e_m\} \subseteq V \times V$. Each edge $e \in E$ is associated with an integer *weight* $w(e) > 0$. We assume that G is *connected* and *simple* (i.e. there exist neither self-loops nor multiple edges). A *forest* is an acyclic subgraph of G, and a *tree* is a connected forest. For a tree T, its weight $w(T)$ is defined as the sum of the weights of constituent edges. A tree is a *spanning tree* if it covers all the nodes of G, and a *minimum spanning tree* is a spanning tree with minimum weight. Throughout the paper, this weight is denoted as $z^\star$, and by MST we denote an arbitrary spanning tree of this weight. (Later we consider spanning trees in some limited framework, where we might have minimum spanning trees with weights larger than $z^\star$. Such a tree is not an MST in our terminology.) Efficient algorithms are well known to find an MST in an undirected graph [1,6,10]. However, there may be more than one MST, and we are concerned with the following problem.

P : List all the MSTs in graph G.

Related to this problem are the algorithms to list all the spanning trees [5,9,11], do the same in non-decreasing order of cost [3,13] and find K-shortest spanning trees in a graph [2,8]. Indeed, P may be solved by finding all the spanning trees, or more preferably by applying a K-shortest

*Corresponding author. Email: yamada@nda.ac.jp

ISSN 0020-7160 print/ISSN 1029-0265 online
© 2010 Taylor & Francis
DOI: 10.1080/00207160903329699
http://www.informaworld.com

spanning tree algorithm with sufficiently large K and truncating its execution as soon as we have a spanning tree of weight larger than $z^\star$. Unfortunately, the latter requires $\mathrm{O}(K)$ memory space which is infeasible for problems with many MSTs. In this article, we give an algorithm, and its refinement, that lists all, and only, MSTs. These algorithms require $\mathrm{O}(m)$ space.

2. A prototype enumeration algorithm

Let $T = \{e^1, \dots, e^{n-1}\}$ be an arbitrary MST in G, which can be obtained by any standard MST algorithms. Then, following the general scheme for solving enumeration problems in combinatorial optimization framework [4,7], we make use of this MST to divide P into the following set of mutually disjoint subproblems ($i = 1, \dots, n-1$).

$P(\{e^1, \dots, e^{i-1}\}, \{e^i\})$: List all the MSTs that contain $e^1, \dots, e^{i-1}$, but do not contain e^i.

More generally, for a forest $F = \{e^1, \dots, e^k\}$ in G and a set of edges $R \subseteq E$ that is disjoint with F (i.e. $F \cap R = \emptyset$), a spanning tree T is (F, R)-*admissible* if it contains all edges of F, but does not contain those of R. That is, T is (F, R)-admissible if and only if $F \subseteq T$ and $R \cap T = \emptyset$. Here we use T to represent a spanning tree, as well as the set of edges included in that tree. F and R are the sets of *fixed* and *restricted* edges, respectively, and we pose the following problem.

$P(F, R)$: List all the MSTs which are (F, R)-admissible.

Clearly, P is identical to $P(\emptyset, \emptyset)$. We note that an (F, R)-admissible spanning tree of the minimum weight can be easily obtained by slightly modifying the standard MST algorithms. Let An_MST(F, R) denote the algorithm to do this, that is it accepts F and R as inputs and returns an (F, R)-admissible spanning tree $T^\star(F, R)$ of the minimum weight $z^\star(F, R)$. Kruskal's or Prim's algorithm [6,10] can be easily tailored for this purpose. If no (F, R)-admissible spanning tree exists, we take the convention of writing $z^\star(F, R) = \infty$.

Then, if $z^\star(F, R) > z^\star$, no MST can be (F, R)-admissible, and hence we *terminate* the subproblem $P(F, R)$. Otherwise, we output $T^\star(F, R)$ as an MST, and make use of this tree to introduce a set of subproblems in the following way. Let $k := |F|$ and the tree be represented as $T^\star(F, R) = F \cup \{e^{k+1}, \dots, e^{n-1}\}$. For $i = k+1, \dots, n-1$, we define

$$F_i := F \cup \{e^{k+1}, \dots, e^{i-1}\}, \quad R_i := R \cup \{e^i\}. \tag{1}$$

Here, in case $i = k+1$, we interpret $\{e^{k+1}, \dots, e^{i-1}\} = \emptyset$ and $F_i = F$. Then, $P(F, R)$ is divided into a set of mutually disjoint subproblems $P(F_i, R_i)$ ($i = k+1, \dots, n-1$), and the problem is solved if we solve all these subproblems. Thus, a prototype algorithm can be constructed in the *divide and conquer* paradigm [12] as follows.

```
ALGORITHM    All_MST(F,R)
Comment: $F$ is a forest in $G$, $R \subseteq E$ and $F \cap R = \emptyset$.

Step 1:  Apply An_MST$(F, R)$ to find $T^\star(F, R)$ and $z^\star(F, R)$.
Step 2:  If $z^\star(F, R) > z^\star$, return.
Step 3:  Output $T^\star(F, R)$, and for $i = k+1, \dots, n-1$ do
             Define $(F_i, R_i)$ by (1), and call All_MST$(F_i, R_i)$.
         endfor.
```

Example 2.1 Consider the graph G of Figure 1, where edge weights are shown in *italic*. We start from $P_0 := P(\emptyset, \emptyset)$, and obtain an MST with $z^\star = 8$ (Step 1). From P_0, 5 children are generated (Step 3) as shown in Figure 2, and in total we obtain 6 MSTs after examining 17 subproblems. Due to the recursive nature of the algorithm, the subproblems are visited and numbered in a *depth-first*

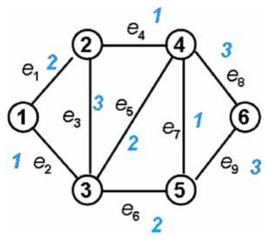

Figure 1. Graph for Example 1.

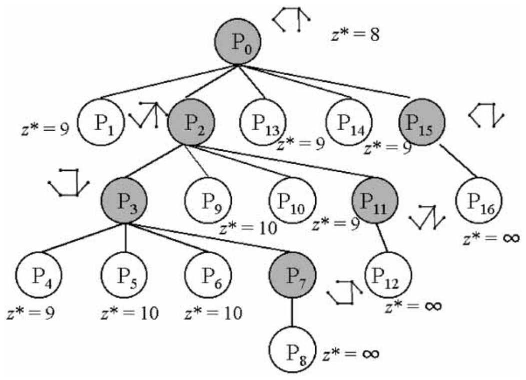

Figure 2. Behaviour of All_MST.

fashion. Here, the subproblems where MSTs are found are shaded with the corresponding MST shown at each shoulder. Details of the subproblems are summarized in Table 1. Here, in each row we show (F, R) that defines subproblem P_i, its *parent*, an (F, R)-admissible minimum spanning tree (if one exists) and the corresponding weight $z^\star(F, R)$ of that tree. Underlined edges represent the set of *newly fixed* (resp. *restricted*) edges at each subproblem, which is denoted as ΔF (resp.

Table 1. Subproblems generated from All_MST.

Subproblems	Parent	F	R	MST	$z^\star(F, R)$
P_0	–	$\emptyset$	$\emptyset$	e_2, e_1, e_4, e_7, e_8	8
P_1	P_0	$\emptyset$	$\underline{e_2}$	–	9
P_2	P_0	$\underline{e_2}$	$\underline{e_1}$	$[e_2], e_5, e_4, e_7, e_8$	8
P_3	P_2	e_2	$e_1, \underline{e_5}$	$[e_2], e_6, e_7, e_4, e_9$	8
P_4	P_3	e_2	$e_1, e_5, \underline{e_6}$	–	9
P_5	P_3	$e_2, \underline{e_6}$	$e_1, e_5, \underline{e_7}$	–	10
P_6	P_3	$e_2, \underline{e_6}, \underline{e_7}$	$e_1, e_5, \underline{e_4}$	–	10
P_7	P_3	$e_2, \underline{e_6}, \underline{e_7}, \underline{e_4}$	$e_1, e_5, \underline{e_9}$	$[e_2, e_6, e_7, e_4], e_8$	8
P_8	P_7	$e_2, e_6, e_7, 4_4$	$e_1, e_5, e_9, \underline{e_8}$	–	∞
P_9	P_2	$e_2, \underline{e_5}$	$e_1, \underline{e_4}$	–	10
P_{10}	P_2	$e_2, \underline{e_5}, \underline{e_4}$	$e_1, \underline{e_7}$	–	9
P_{11}	P_2	$e_2, \underline{e_5}, \underline{e_4}, \underline{e_7}$	$e_1, \underline{e_8}$	$[e_2, e_5, e_4, e_7], e_9$	8
P_{12}	P_{11}	e_2, e_5, e_4, e_7	$e_1, e_8, \underline{e_9}$	–	∞
P_{13}	P_0	$\underline{e_2}, \underline{e_1}$	$\underline{e_4}$	–	9
P_{14}	P_0	$\underline{e_2}, \underline{e_1}, \underline{e_4}$	$\underline{e_7}$	–	9
P_{15}	P_0	$\underline{e_2}, \underline{e_1}, \underline{e_4}, \underline{e_7}$	$\underline{e_8}$	$[e_2, e_1, e_4, e_7], e_9$	8
P_{16}	P_{15}	e_2, e_1, e_4, e_7	$e_8, \underline{e_9}$	–	∞

ΔR) for later use. In the column 'MST', edges within braces are the fixed edges, and hyphen(-) shows that the subproblem is terminated because $z^\star(F, R) > z^\star$.

The computational complexity of the above algorithm can be evaluated as follows. Let N denote the number of MSTs included in G, and $T_{\mathrm{MST}}(n, m)$ be the time required to solve an MST problem for a graph with n nodes and m edges. Note that each subproblem, including an MST, produces at most $n-1$ children. So the total number of subproblems generated in All_MST is at most $Nn - N + 1 \approx Nn$. In each subproblem, we solve an MST problem, which requires $O(T_{\mathrm{MST}}(n, m))$ time. Thus, the total time complexity is $\mathrm{O}(NnT_{\mathrm{MST}}(n, m))$.

To access the space complexity, consider the tree of subproblems as shown in Figure 2. We note that the maximum height of the tree is at most m, since at each branch at least one edge is either fixed or restricted. While solving $P(F, R)$, we only need to keep the differential information $(\Delta F', \Delta R')$ in memory for all ancestors $P(F', R')$ of $P(F, R)$. From these incremental information, we can reconstruct (F, R), and to keep these in memory $\mathrm{O}(m)$ space suffices. Thus, we have the following.

THEOREM 2.2 *The time and space complexities of All_MST are $O(NnT_{MST}(n, m))$ and $O(m)$, respectively.*

Remark 2.3 A simple implementation of Kruskal's algorithm [6] runs in $T_{\mathrm{MST}}(n, m) = \mathrm{O}(mn)$ time, but by using sophisticated data structure such as Fibonacci heap, this can be improved to $T_{\mathrm{MST}}(n, m) = \mathrm{O}(m + n\log n)$ [1]. Therefore, the time complexity of All_MST is

$$\mathrm{O}(N(mn + n^2 \log n)). \tag{2}$$

3. A cut-set-based algorithm

Let T be an MST of G, and e an arbitrary edge of T. Deleting e from T divides the tree into two connected components with vertexes sets, say V_1 and V_2. We introduce the *cut-set* induced by $e \in T$ as the set of edges with one endpoint in V_1 and the other in V_2. If we define this as $\mathrm{Cut}(e) := \{e' \in E \mid e' \in (V_1 \times V_2) \cup (V_2 \times V_1)\}$, the following Proposition is clear from the 'cut optimality condition' [1, Theorem 13.1] for MST.

PROPOSITION 3.1 *Let T be an MST and $e \in T$. Then, for an arbitrary edge $e' \in \mathrm{Cut}(e)$,*

$$w(e') \geq w(e). \tag{3}$$

That is, $e \in T$ is an edge of the minimum weight in $\mathrm{Cut}(e)$. For a pair of edges $e \in T$ and $e' \in \mathrm{Cut}(e)\backslash\{e\}$, $T \cup \{e'\}\backslash\{e\}$ defines another spanning tree, which is denoted as $T \cup e'\backslash e$ for simplicity. Let $G^{(e)}$ denote the graph obtained from G by deleting e. We then have the following.

THEOREM 3.2 *Let T be an MST, $e \in T$ and $e' \neq e$ be* second minimum *in weight in* $\mathrm{Cut}(e)$. *Then, $T' := T \cup e'\backslash e$ is a minimum spanning tree in $G^{(e)}$.*

Proof Suppose that T' is not a minimum spanning tree in $G^{(e)}$. Then, by the 'path optimality condition' [1, Theorem 13.3] there exists a pair of edges $c \notin T'$ and $d \in T'$ such that (1) $T'' := T' \cup c\backslash d$ is a spanning tree and (2) $w(c) < w(d)$. We note that $c \in \mathrm{Cut}(e)$, otherwise $\tilde{T} := T \cup c\backslash d$ is a spanning tree with $w(\tilde{T}) < z^\star$, contradicting that T is an MST. Thus, we have

$$w(e) \leq w(e') \leq w(c) < w(d). \tag{4}$$

Let the cycles included in $T^\star \cup \{e'\}$, $T^\star \cup \{c\}$ and $T' \cup \{c\}$ be C_0, C_1 and C_2, respectively. Since $C_2 = C_0 \oplus C_1$ (in Boolean sense) and $d \in C_2$, we have $d \in C_0$ or $d \in C_1$. Define

$$\hat{T} := \begin{cases} T^\star \cup e' \backslash d & \text{if } d \in C_0, \\ T^\star \cup c \backslash d & \text{if } d \in C_1. \end{cases} \tag{5}$$

Then, $\hat{T}$ is a spanning tree with $w(\hat{T}) < z^\star$, which is a contradiction. ■

The following Corollaries are easily proved.

Corollary 3.3 *Let T be an MST, $e \in T$ and $e' \neq e$ be second minimum in weight in* Cut(e). *If $w(e) = w(e')$, then $T \cup e' \backslash e$ is an MST; otherwise, no MST exists in $G^{(e)}$.*

Corollary 3.4 *If all the edges of G are of distinct weights, then MST is unique.*

Let T be an MST and $e \in T$. We call $\tilde{e} \in \text{Cut}(e)$ a *substitute* of e if $\tilde{e} \neq e$ and $w(\tilde{e}) = w(e)$. $S(e)$ stands for the set of all substitutes of e, that is,

$$S(e) := \{\tilde{e} \in \text{Cut}(e) | \tilde{e} \neq e, w(\tilde{e}) = w(e)\}. \tag{6}$$

On the basis of the above theorem, we can modify All_MST in the following way. At each subproblem, in addition to the pair of sets (F, R) representing the fixed and restricted edges, we assume that an (F, R)-admissible MST exists. Let this tree be T. In the following algorithm, in the subproblem $P(F, R)$, for each $e^i \in T \backslash F$, we look for a substitute $\tilde{e}^i \in S(e)$. If this is found, we obtain $T \cup \tilde{e}^i \backslash e^i$ as a new MST, and use this to generate a sub-subproblem. Then, the algorithm is described as follows.

Algorithm *All_MST*$_1(F, R, T)$
Comment: T is an (F, R)-admissible MST, which is written as $T = F \cup \{e^{k+1}, \ldots, e^{n-1}\}$, with $F = \{e^1, \ldots, e^k\}$.

```
Step 1: For $i = k + 1, \ldots, n - 1$, do the following:
        • Find the cut-set Cut$(e^i)$.
        • Find, if one exists, a substitute $\tilde{e}^i \in S(e^i)$.
Step 2: For $i = k + 1, \ldots, n - 1$, if $\tilde{e}_i$ exists do the following:
        • Set $T_i := T \cup \tilde{e}^i \setminus e^i$ and output $T_i$. {Comment: A new MST is found}
        • Set $F_i := F \cup \{e^{k+1}, \ldots, e^{i-1}\}$ and $R_i := R \cup \{e^i\}$.
        • Call All_MST$_1(F_i, R_i, T_i)$ recursively.
```

Example 3.5 For the graph of Figure 1, Figure 3 shows the behaviour of All_MST$_1$, where details of the generated subproblems are summarized in Table 2. In addition to the information given in Table 1, here is a substitute for the newly restricted edges (ΔR) in each subproblem. In this algorithm, each subproblem generated includes an (F, R)-admissible MST as shown in Figure 3, and newly fixed edges (ΔF) are underlined in Table 2.

For the computational complexity of this algorithm, we have the following.

Theorem 3.6 *The running time of All_MST*$_1$ *is $O(Nmn)$.*

Proof Note that Cut(e^i) can be found in O(m) time. At each subproblem, this is repeated at most n times. ■

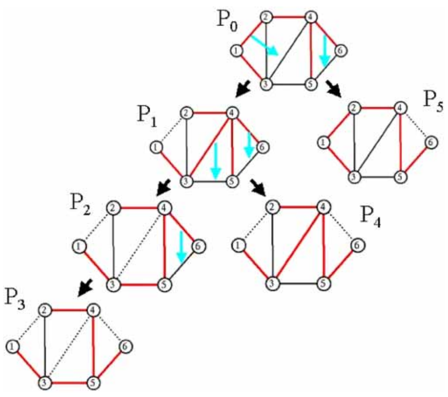

Figure 3. Behaviours of All_MST$_1$.

Table 2. Subproblems generated for the graph of Figure 1.

Subproblems	Parent	F	R	MST	Substitute
P_0	–	$\emptyset$	$\emptyset$	e_2, e_1, e_4, e_7, e_8	–
P_1	P_0	$\underline{e_2}$	$\underline{e_1}$	$[e_2], e_5, e_4, e_7, e_8$	e_5
P_2	P_1	e_2	$e_1, \underline{e_5}$	$[e_2], e_6, e_4, e_7, e_8$	e_6
P_3	P_2	$e_2, \underline{e_6}, \underline{e_4}, \underline{e_7}$	$e_1, e_5, \underline{e_8}$	$[e_2, e_6, e_4, e_7], e_9$	e_9
P_4	P_1	$e_2, \underline{e_5}, \underline{e_4}, \underline{e_7}$	$e_1, \underline{e_8}$	$[e_2, e_5, e_4, e_7], e_9$	e_9
P_5	P_0	$\underline{e_2}, \underline{e_1}, \underline{e_4}, \underline{e_7}$	$\underline{e_8}$	$[e_2, e_1, e_4, e_7], e_9$	e_9

4. A further improvement

In the previous section, it took O(m) time, in the worst case, to find a substitute $\tilde{e}^i$ for each $e^i \in T$. This computation was carried out for each e^i from scratch, and the computation of all substitutes took O(mn) time. In this section, we present an algorithm that finds $\tilde{e}^i$ one-by-one, from the reduced set of possible cut-set edges for e^i, and after completing step i this information is carried on to the next step. We try to make the size of this set as small as possible, and as a result, in total we obtain the set of all substitutes $\{\tilde{e}^i \mid e^i \in T \backslash F\}$ in O$(m \log n)$ time, instead of O(mn) in All_MST$_1$.

To accomplish this, at each subproblem $P(F, R)$ with an (F, R)-admissible MST T, we renumber vertexes in *postorder* fashion [12] as we traverse T from an arbitrary 'root' vertex. Let the vertexes thus renumbered be $\{v^i | i = 1, \ldots, n\}$. Then, T is a tree rooted at v^n, and by e^i we denote the tree edge connecting v^i to its parent vertex. (See Figure 4, where T is shown in bold). Associated with v^i is an interval $[\underline{\sigma}_i, \overline{\sigma}_i]$, which represents the set of descendants of this vertex, that is,

$$j \in [\underline{\sigma}_i, \overline{\sigma}_i] \Leftrightarrow v^j \text{ is a descendant of } v^i \text{ in tree } T \text{ rooted at } v^n. \tag{7}$$

Let $E^i := \{(v^i, v^j) \in E | (v^i, v^j) \notin T\}$ be a set of non-tree edges incident on node v^i, and Q be a set of elements of the form $(w, v, v') \in Z \times V \times V$. Here, $(w, v, v') \in Q$ means that the edge $e = (v, v') \in E$ has weight $w(e) = w$, and is a *candidate* of a cut-set edge at this, and subsequent, stage. We call Q the set of *quasi-cuts*, and assume that it is *lexicographically* ordered with respect to w and v. For $e^i \in T \backslash F$, we make use of Q to find its substitute $\tilde{e}^i$ and update it for the next i in the following way.

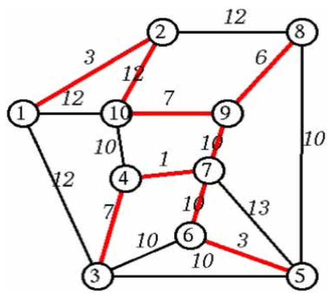

Figure 4. Graph for Example 3.

ALGORITHM *Substitute*(F, R, T)

```
Step 1: Set $Q := \emptyset$.
Step 2: For $i = 1$ to $n - 1$ do the following.
        (1) For $\forall e = (v^i, v^j) \in E^i \setminus R$ do either of the following.
             a) If $j < \underline{\sigma}_i$, {Comment: Reverse the direction.}
                 • If $(w(e), j, i) \in Q$, delete it from $Q$.
                 • Insert $(w(e), i, j)$ into $Q$.
            (b) If $j \in [\underline{\sigma}_i, \overline{\sigma}_i]$,
                 • If $(w(e), j, i) \in Q$, delete it from $Q$.
            (c) If $j > \overline{\sigma}_i$,
                 • Insert $(w(e), i, j)$ into $Q$.
        (2) If $e^i \notin F$ do the following.
            (a) Find $(w, i', j') \in Q$ such that $w = w(e^i)$ and $i' \in [\underline{\sigma}_i, \overline{\sigma}_i]$
            (b) If such an $(w, i', j')$ is found with $j' \in [\underline{\sigma}_i, \overline{\sigma}_i]$,
                 • Delete $(w, i', j')$ from $Q$, and go to (2)-(a).
            (c) If such an $(w, i', j')$ is found with $j' \notin [\underline{\sigma}_i, \overline{\sigma}_i]$,
                 • Set $\tilde{e}^i := (v^{i'}, v^{j'})$. {Comment: Substitute for $e^i$ found.}
```

In Step 2-(1), we update Q by including the edges of $E^i \backslash R$ as a candidate of cut-set edges for e^i and subsequent tree edges. Specifically, in Step 2-(1)-(c) edge (v^i, v^j) is included in Q if this is an edge from v^i to v^j with $j > i$. On the other hand, edges from e^i to its descendants are deleted [Step 2-(1)-(b)], since from now on these can no longer be a cut-set edge. If we have an edge coming into v^i from a previously found vertex v^j, the direction is reversed [Step 2-(1)-(a)].

Next, in Step 2-(2) we look for a substitute $\tilde{e}^i$ of e^i included in Q. If we have an edge with the weight identical to $w(e^i)$ and emerging from a vertex within the interval $[\underline{\sigma}_i, \overline{\sigma}_i]$ [Step 2-(2)-(c)], this is a substitute for e^i, and thus we are done. If this is an edge from the above interval to the same interval, we delete this edge from Q, and repeat Step 2-(2) again.

Example 4.1 Consider the graph of Figure 4 with an MST T shown in thick lines. Nodes and edges are numbered in the postorder fashion as traversed along T from node v^{10}. Table 3 shows $w(e^i)$, $[\underline{\sigma}_i, \overline{\sigma}_i]$ and Q at each stage. The elements superscribed with $\circ$, $\times$ and ! show, respectively, the newly found, deleted and substitute for e^i. The elements associated with † shows that the edge direction is reversed here.

Let All_MST$_2$ be the algorithm obtained from All_MST$_1$ by replacing Step 1 as follows.
Step 1†: Execute Substitute(F, R, T).

Table 3. Behaviour of *Substitute* for the graph of Figure 4.

i	$w(e^i)$	$[\underline{\sigma}_i, \overline{\sigma}_i]$	Q	i	$w(e^i)$	$[\underline{\sigma}_i, \overline{\sigma}_i]$	Q
1	3	[1,1]	(12:1, 3)$^{\circ}$	6	10	[5,6]	(10:4, 10)
			(12:1, 10)$^{\circ}$				(10:5, 3)$^{!}$
							(10:5, 8)
2	12	[1,2]	(12:1, 3)$^{!}$				(10:6,3)†
			(12:1,10)				(12:1, 10)
			(12:2, 8)$^{\circ}$				(12:2, 8)
							(12:3, 1)
3	7	[3,3]	(10:3, 5)$^{\circ}$				(13:5, 7)
			(10:3, 6)$^{\circ}$				
			(12:1, 10)	7	10	[3,7]	(10:4, 10)$^{!}$
			(12:2, 8)				(10:5, 3)$^{\times}$
			(12:3, 1)†				(10:5, 8)
							(10:6, 3)$^{\times}$
4	1	[3,4]	(10:3, 5)				(12:1, 10)
			(10:3, 6)				(12:2, 8)
			(10:4,10)$^{\circ}$				(12:3, 1)
			(12:1,10)				
			(12:2, 8)	8	6	[8,8]	(10:4,10)
			(12:3, 1)				(10:8, 5)†
							(12:1,10)
5	3	[5,5]	(10:3, 6)				(12:3, 1)
			(10:4,10)				(12:8, 2)†
			(10:5, 3)†				
			(10:5, 8)$^{\circ}$	9	7	[3,9]	(10:4,10)
			(12:1,10)				(10:8, 5)
			(12:2, 8)				(12:1, 10)
			(12:3, 1)				(12:3, 1)
			(13:5, 7)$^{\circ}$				(12:8, 2)

Computational complexity of All_MST$_2$ can be evaluated as follows. First, note that Q includes at most m elements, and for each non-tree edges at most two *Insert*s and two *Delete*s are executed. For each tree edge at most one *Find* is executed. Since Q is an ordered set, each of *Insert*, *Delete* and *Find* can be done in O$(\log m)$ time, provided that Q is organized as an appropriate binary tree (such as the Adelson-Velskii and Landis tree (AVL-tree) [12]). Thus, in total *Substitutes* can be done in O$(m \log m) =$ O$(m \log n)$ time. All other computations, such as traversing G along T, renumbering V in postorder fashion and finding intervals $[\underline{\sigma}_i, \overline{\sigma}_i]$, can be accomplished in O(m) time. Thus, we have the following.

THEOREM 4.2 *The running time of All_MST$_2$ is $O(Nm \log n)$.*

5. Numerical experiments

We have implemented All_MST and All_MST$_1$ in C language and conducted a series of numerical experiments on an IBM RS/6000 44P MODEL 270 workstation (375 MHz). Throughout the experiments, All_MST and All_MST$_1$ found the same number of MSTs.

5.1 *Complete graphs with constant edge weights*

First, we consider the complete graph K_n with edge weights fixed at $w(e) \equiv 1$ for all $e \in E$. In this case, all the spanning trees are MSTs, and the total number of spanning trees can be shown [9] to be

$$N = n^{n-2}. \tag{8}$$

Table 4. Result of experiments (complete graphs, constant edge weights).

		All_MST		All_MST$_1$
Graph	N	#sub	CPU	CPU
K_3	3	6	0.00	0.00
K_4	16	29	0.00	0.00
K_5	125	212	0.00	0.00
K_6	1296	2117	0.02	0.01
K_7	16,807	26,830	0.22	0.08
K_8	262,144	412,015	3.60	1.35
K_9	4,782,969	7,433,032	60.74	26.99
K_{10}	100,000,000	154,076,201	1694.27	616.10

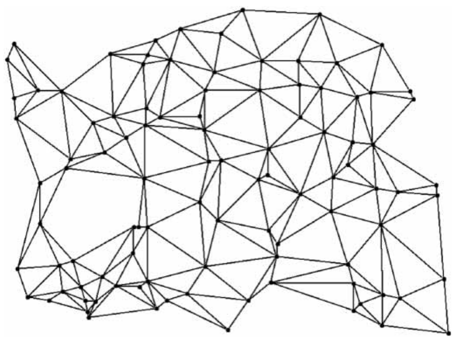
Figure 5. Planar graph $P_{100\times 260}$.

Table 4 gives the result of experiments for K_n. The number of spanning trees (N), the number of generated subproblems (#sub) and CPU time in seconds are shown. Both All_MST and All_MST$_1$ compute N_n correctly for $n \leq 10$, and All_MST$_1$ is approximately 2.5 times faster than All_MST.

5.2 *Planar graphs with random edge weights*

Let $P_{n\times m}$ be a planar graph with n nodes and m edges (See Figure 5 for $P_{100\times 260}$), and the edge weights are both randomly and uniformly distributed over $[1, 10^L]$ ($L = 2, 3$).

Table 5 summarizes the result of the experiments for these graphs. Each row is the average of 10 independent runs. In this case, All_MST$_1$ runs 30 – 60 times faster than All_MST. The number of MSTs is much smaller in the case where edge weights are distributed over [1, 1000] ($L = 3$) than in the case $L = 2$. This is explained as follows. In the latter case, the chance of having edges of identical weights are much larger than in the former case.

5.3 *Complete graphs with random edge weights*

Table 6 gives the result for complete graphs with random edge weights uniformly distributed over $[1, 10^L]$ ($L = 2, 3$). Again each row is the average of 10 independent runs. In this case, All_MST$_1$ is faster than All_MST, but only 2–4 times, as opposed to 50–70 times in the case of planar graphs. This may be explained by considering the ratio of the number of subproblems generated in All_MST over the total number of MSTs (i.e. #sub/N in Tables 5 and 6). For planar graphs this ratio is approximately 20–50 in the case of $L = 2$ and 100–500 for $L = 3$. Similarly, for complete graphs the ratio is 3–10 ($L = 2$) and 10–40 ($L = 3$), respectively. Thus, the ratio is larger for planar graphs than for complete graphs, meaning that All_MST generates more

Table 5. Result of experiments (planar graphs, random edge weights).

				All_MST		All_MST_1
L	Graph	$z^\star$	N	# sub	CPU	CPU
2	$P_{20\times 46}$	459.1	1.2	20.6	0.00	0.00
	$P_{50\times 127}$	1110.9	1.2	53.2	0.01	0.00
	$P_{100\times 260}$	2224.7	2.2	119.3	0.04	0.00
	$P_{200\times 560}$	4211.6	18.0	1108.3	1.34	0.06
	$P_{400\times 1120}$	8367.9	147.0	6995.3	29.68	0.85
	$P_{600\times 1680}$	12,615.5	1921.6	65,154.7	587.30	14.16
	$P_{800\times 2240}$	17,065.8	15,592.8	603,647.5	5604.70	185.15
3	$P_{20\times 46}$	4080.5	1.0	20.0	0.00	0.00
	$P_{50\times 127}$	10,535.3	1.1	52.9	0.01	0.00
	$P_{100\times 260}$	21,479.5	1.1	107.4	0.04	0.00
	$P_{200\times 560}$	40,590.2	1.6	239.0	0.30	0.01
	$P_{400\times 1120}$	81,766.4	1.9	518.8	2.24	0.07
	$P_{600\times 1680}$	122,550.5	3.1	1116.5	10.46	0.23
	$P_{800\times 2240}$	162,966.2	2.0	1169.3	20.83	0.36
	$P_{1000\times 2800}$	204,548.9	3.2	1673.4	43.28	0.67

Table 6. Result of experiments (complete graphs, random edge weights).

				All_MST		All_MST_1
L	Graph	$z^\star$	N	# sub	CPU	CPU
2	K_{20}	120.9	3.6	38.2	0.00	0.00
	K_{40}	141.1	11.0	112.8	0.02	0.01
	K_{60}	152.5	1669.2	8840.1	3.15	1.57
	K_{80}	165.9	1,494,553.0	4,647,056.0	3236.55	1511.29
3	K_{20}	1114.2	1.1	20.2	0.00	0.00
	K_{40}	1224.0	2.5	59.9	0.01	0.01
	K_{60}	1260.5	4.1	101.8	0.04	0.02
	K_{80}	1253.0	3.1	136.9	0.10	0.05
	K_{100}	1264.1	9.3	227.4	0.26	0.12
	K_{120}	1251.6	14.3	361.4	0.59	0.28
	K_{140}	1281.3	123.2	1768.9	3.78	1.88
	K_{160}	1274.3	337.4	7966.7	24.49	11.46
	K_{180}	1288.8	3980.0	63,892.2	287.48	136.65
	K_{200}	1296.4	7434.4	145,946.9	871.92	427.49

'redundant' subproblems in planar graphs than in complete graphs. Since All_MST_1 does not produce such an redundant subproblem, All_MST_1 is more superior to All_MST in planar graphs than in complete graphs.

6. Conclusion

We have developed an algorithm to list all the minimum spanning trees in an undirected graph, and explored some properties of cut-sets. Furthermore, based on these, we have constructed an improved algorithm, which runs in $O(Nm \log n)$ time and $O(m)$ space. We have not implemented All_MST_2, but this is expected to be better than All_MST or All_MST_1 for larger problems. Numerical experiment of this part is left for our future work.

Acknowledgements

The authors are grateful to two anonymous referees and and a Editor for their constructive criticisms and suggestions. Numerous comments from Professor Kazuo Ouchi were substantial in improving the quality of English in this paper, for which we express deep appreciation.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.
[2] P.M. Camerini, L. Fratta, and F. Maffioli, *Efficient methods for ranking trees*, Proceedings of the 3rd International Symposium on Network Theory, Split, Yugoslavia, 1975, pp. 1–10.
[3] H.N. Gabow, *Two algorithms for generating weighted spanning trees in order*, SIAM J. Comput. 6 (1977), pp. 139–150.
[4] H.W. Hamacher and M. Queyranne, *K-best solutions to combinatorial optimization problems*, Ann. Oper. Res. 4 (1985), pp. 123–143.
[5] S. Kapoor and H. Ramesh, *Algorithms for enumerating all spanning trees of undirected and weighted graphs*, SIAM J. Comput. 24 (1995), pp. 247–265.
[6] J.B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. Am. Math. Soc. 7 (1956), pp. 8–50.
[7] E.L. Lawler, *A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem*, Manag. Sci. 18 (1972), pp. 401–405.
[8] T. Matsui, *A flexible algorithm for generating all the spanning trees in undirected graphs*, Algorithmica, 18 (1997), pp. 530–543.
[9] G.J. Minty, *A simple algorithm for listing all the trees of a graph*, IEEE Trans. Circuit Theory CT-12 (1965), p. 120.
[10] R.C. Prim, *Shortest connection networks and some generalizations*, Bell Syst. Tech. J. 36 (1957), pp. 1389–1401.
[11] R.C. Read and R.E. Tarjan, *Bounds on backtrack algorithms for listing cycles, paths, and spanning trees*, Networks 5 (1975), pp. 237–252.
[12] R. Sedgewick, *Algorithms in C*, 3rd ed., Addison-Wesley, Reading, MA, 1998.
[13] K. Sörensen and G.K. Janssens, *An algorithm to generate all spanning trees of a graph in order of increasing cost*, Pesqui. Oper., 25 (2005), pp. 219–229.