Information Processing Letters 39 (1991) 277–281
North-Holland

13 September 1991

Finding the most vital edge with respect to minimum spanning tree in weighted graphs

Lih-Hsing Hsu, Rong-Hong Jan, Yu-Che Lee and Chun-Nan Hung

Department of Information and Computer Science, National Chiao Tung University, Hsinchu 30050, Taiwan, ROC

Maw-Sheng Chern

Department of Industrial Engineering, National Tsing Hua University, Hsinchu 30043, Taiwan, ROC

Communicated by K. Ikeda
Received 7 January 1991
Revised 1 April 1991

Keywords: Data structures, design of algorithms, minimum spanning trees

1. Introduction

In many applications, the network designer may want to know which edges in the network are most important to him. If these edges are removed from the network, there will be a great decrease in its performance. Such edges are called the most vital edges in a network. Several papers [1,2] have been presented to find the most vital edges. However, they are only concerned with the effect of the maximum flow or the shortest path in the network. In this paper, we will consider the effect of a minimum spanning tree in the network.

Most graph-theoretic terms used in this paper are standard (e.g., [3]). Here, we limit ourselves to defining the most commonly used terms and those that may produce confusion. $G=(V, E)$ is called a *graph* if V is a finite set and E is a subset of $\{(a, b) \mid a \neq b, (a, b)$ is an unordered pair of $V\}$. We say V is the *vertex set* of G, E is the *edge set* of G. Let $p=|V|$ and $q=|E|$. Let $\bar{E}$ be a subset of E. We use $G-\bar{E}$ to denote the graph $G'=(V, E-\bar{E})$. In particular, we use $G-e$ and $G+e$ to denote the graph $G-\{e\}$ and $G+\{e\}$, respectively. Graph $H=(V', E')$ is called a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. A subgraph $H=(V', E')$ of G with $V'=V$ is called a *spanning subgraph* of G. A *spanning tree* of G is a connected spanning subgraph of G that contains no cycles.

A *weighted graph* is a graph $G=(V, E)$ with a weight $w(e)$ assigned to every edge e in E. The *weight* of a spanning tree T, $w(T)$, is defined to be the summation of $w(e)$ for all e in T. A spanning tree T in G is called a *minimum spanning tree* if $w(T) \leqslant w(T')$ for all spanning trees T' in G. Minimum spanning trees have many applications in network design, VLSI, geometric optimization and so on. There are two best-known algorithms used in finding the minimum spanning tree in a weighted graph. One is Kruskal's algorithm [4] and the other is Prim's algorithm [5]. It is known that Kruskal's algorithm takes $O(q \log q)$ time whereas Prim's algorithm takes $O(p^2)$ time.

Let $g(G)$ denote the weight of a minimum spanning tree of G if G is connected; otherwise, $g(G)=\infty$. An edge e is called a *most vital edge* (MVE) in G if $g(G-e) \geqslant g(G-e')$ for every edge e' of G. The problem of finding such an edge is called the 1-MVE problem. Two al-

gorithms with time complexities $O(q \log q)$ and $O(p^2)$ are presented for solving the 1-MVE problem in this paper.

2. Some interesting properties

Throughout this paper, we assume that the costs of edges in G are different. With this assumption, the minimum spanning tree in G is unique. A naive way to find the most vital edge in G is finding $g(G-e)$ for each edge e in E by applying Kruskal's algorithm. The most vital edge in G is thus easily obtained. However, this method takes $O(q^2 \log q)$ time. We may reduce the time to $O(pq \log q)$ once we have the following lemma:

Lemma 1. *Let T_G be the minimum spanning tree of G. Then, the most vital edge of G is one of the edges in T_G.*

Proof. It is clear that $g(G-e) > g(G)$ if the edge e is in T_G. Let e^* be the most vital edge. If e^* is not in T_G, then T_G is a spanning tree of $G-e^*$. This implies that $g(G-e^*) = g(G)$. There is a contradiction. Hence, the most vital edge of G must be one of the edges in T_G. □

With Lemma 1, the most vital edge of G is in its minimum spanning tree T_G. Thus, only $|T_G| = p-1$ edges are considered and the time complexity of the above naive method can be reduced to $O(pq \log q)$. We need the following discussion to further reduce the time complexity.

Let Ω be the set of all edges in T_G and $B = E - \Omega$. Let $e = (a, b)$ be an edge in B and $a = v_0, v_1, \ldots, v_k = b$ be the path with endpoints a and b in T_G. For each $e = (a, b)$, we define an edge set R_e that contains all edges in path $a = v_0, v_1, \ldots, v_k = b$. Then for $e' \in \Omega$, we define the edge sets $\overline{R}_{e'}$ as containing all edges e such that $e' \in R_e$. Let $f(e')$, where $e' \in \Omega$, be the edge e in B such that $w(e) = \min\{w(e'') \mid e'' \in \overline{R}_{e'}\}$. For example,

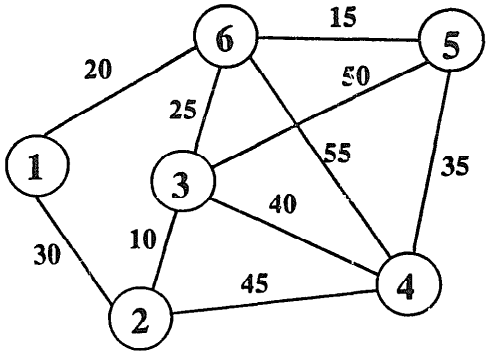

Fig. 1. Graph G.

the bold edges in Fig. 2 show the minimum spanning tree of the graph G given in Fig. 1. Then,

$R_{(1,2)} = \{(2, 3), (3, 6), (6, 1)\}$,

$R_{(3,4)} = \{(3, 6), (6, 5), (5, 4)\}$,

$R_{(2,4)} = \{(2, 3), (3, 6), (6, 5), (5, 4)\}$,

$R_{(3,5)} = \{(3, 6), (6, 5)\}$

and

$R_{(4,6)} = \{(4, 5), (5, 6)\}$.

Since $(2, 3) \in R_{(1,2)}$ and $R_{(2,4)}$, the edge set $\overline{R}_{(2,3)} = \{(1, 2), (2, 4)\}$. Obviously, $f((2, 3)) = (1, 2)$.

Lemma 2. *If e is in Ω, then $T_{G-e} = T_G - e + f(e)$. Hence $w(T_{G-e}) = w(T_G) - w(e) + w(f(e))$.*

Proof. The proof is obtained by applying Kruskal's algorithm on G and $G-e$. Without loss of generality, $w(e_1) < w(e_2) < \cdots < w(e_q)$ is assumed. Kruskal's algorithm [4] initializes the spanning

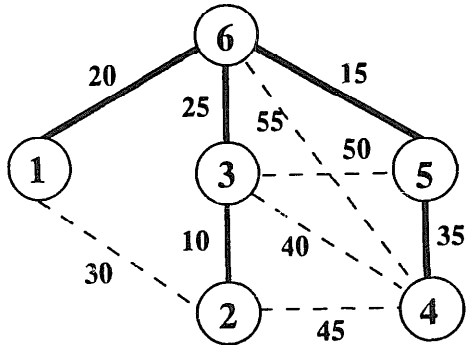

Fig. 2. Minimum spanning tree of the graph G in Fig. 1.

forest F_0 of G with no edge. At each iteration i, check if $F_{i-1} \cup \{e_i\}$ is acyclic or not. If $F_{i-1} \cup \{e_i\}$ is acyclic, then set $F_i = F_{i-1} \cup \{e_i\}$. Otherwise, set $F_i = F_{i-1}$.

Let F_i and F_i' be the intermediate spanning forests obtained by applying Kruskal's algorithm on G and $G - e$, respectively. Let $e = e_j$ and $f(e) = e_h$. Then $j < h$, otherwise e_h is an edge of T_G. Thus, $F_k' = F_k$ if $k < j$. Since e_h is the edge with $w(e_h) = \min\{w(e'') \mid e'' \in \bar{R}_e\}$, we have $F_k' = F_k - e$, if $j \leqslant k < h - 1$; and $F_k' = F_k - e + f(e)$ if $k \geqslant h - 1$. Hence $T_{G-e} = T_G - e + f(e)$. □

The edge $f(e)$ is called the *entering edge* with respect to the *leaving edge* e. With Lemma 2, we may easily compute the most vital edge once we know the entering edge $f(e)$, for every $e \in \Omega$. In the following, there are some properties about the trees that will be used later. Let T be a spanning tree of G. We may pick any vertex of T as a root and label the vertices of T from 1 to p in postorder. Thus, each vertex can be identified by its postordered number. All edges of T can be written as $(v, p(v))$, where v is a nonroot node and $p(v)$ is the parent of v. Let us regard every vertex as a descendant of itself. Then, we have Lemma 3.

Lemma 3. *If $e = (v, p(v))$ is an edge of T, then $f(e)$ must be an edge joining a vertex of a descendant of v to a nondescendant of v.*

3. An O($q \log q$) time algorithm for the 1-MVE problem

The main idea of the algorithm presented in this section is to find the leaving edge set $f^{-1}(e')$ for every $e' \in E - \Omega$. Then, we apply Lemma 2 to compute the most vital edge in G.

Algorithm 1

Step 1. Apply Kruskal's algorithm to find the minimum spanning tree T_G of $G = (V, E)$. Let Ω be the set of edges in T_G. Sort the edges in set $B = E - \Omega = \{e_1, e_2, \ldots, e_{q-p+1}\}$ in terms of their weights and $w(e_1) < w(e_2) < \cdots < w(e_{q-p+1})$ is assumed. Let $(v_s, u_s) = e_s$.

Step 2. Find the leaving edge set $f^{-1}(e_s), f^{-1}(e_s) \subset \Omega$, for entering edges e_s, $s = 1, 2, \ldots, q - p + 1$, as follows.

2.1. Let $F_0 = \emptyset$. Let R_{e_s} be the set of all edges in the unique path from the vertex v_s to u_s in T_G.

2.2. Set $f^{-1}(e_s) = R_{e_s} - F_{s-1}$ and $F_s = F_{s-1} \cup f^{-1}(e_s)$, $s = 1, 2, \ldots, q - p + 1$.

Step 3. For each edge e in T_G, compute $w(T_{G-e}) = w(T_G) - w(e) + w(f(e))$. Find the edge e^* such that $w(T_{G-e^*}) = \max\{w(T_{G-e}) \mid \text{the edge } e \text{ is in } T_G\}$.

By Lemma 2, it is clear that $f(e) = e_1$ for all edges e in the unique path from the vertex v_1 to u_1 in T_G. In general, $f(e) = e_s$ for all edges e in the unique path from the vertex v_s to u_s but not in $\bigcup_{j=1}^{s-1} f^{-1}(e_j)$. Thus, Step 2 finds $f^{-1}(e_s)$ correctly. We illustrate Algorithm 1 via the example in Fig. 1. Step 1 finds the minimum spanning tree T_G shown in Fig. 2. In Step 2, we find the leaving edge sets $f^{-1}(e_s)$, for entering edges

$$(e_1, e_2, \ldots, e_{q-p+1}) = ((1, 2), (3, 4), (2, 4), (3, 5), (4, 6))$$

as follows:

$$f^{-1}((1, 2)) = \{(2, 3), (3, 6), (1, 6)\},$$

$$f^{-1}((3, 4)) = \{(4, 5), (5, 6)\},$$

$$f^{-1}((2, 4)) = \emptyset,$$

$$f^{-1}((3, 5)) = \emptyset,$$

and

$$f^{-1}((4, 6)) = \emptyset.$$

Step 3 computes

$$\big(w(T_{G-(2.3)}), w(T_{G-(3.6)}), w(T_{G-(1.6)}), w(T_{G-(4.5)}), w(T_{G-(5.6)})\big) = (125, 110, 115, 110, 130)$$

and then finds the MVE $e^* = (5, 6)$.

Now, we describe the time complexity of Algorithm 1. Obviously, Steps 1 and 3 take

$O(q \log q)$ and $O(p)$ time, respectively. At Step 2.1, the unique path from vertex v_s to u_s can be determined as follows. Let vertex v_r be the root of T_G. Define the *depth* of a vertex v, denoted as $l(v)$, in T_G is the distance of v from the root v_r. (All edges have distance 1.) Search from vertex v_s and u_s to root v_r, respectively. Start with the greater depth vertex and stop when a common ancestor v_k is found. Then the unique path from the vertex v_s to u_s is obtained. With this implementation, Step 2 takes $O(pq)$ time. Thus, the total time for Algorithm 1 is $O(pq)$. However, we may use the data structure UNION-and-FIND [6] on disjoint sets to reduce the time to $O(q \log q)$.

Step 2 of Algorithm 1 can be modified as follows by introducing the operations FIND(i) and UNION(i, j). The operation of FIND(i) is to determine the root of the tree containing element i. UNION(i, j) requires two trees with roots i and j to be joined and assigns the vertex with the smaller depth among i and j as the root of the resultant tree.

Step 2. For $s = 1, 2, \ldots, q - p + 1$, use the following procedure to find the leaving edge set $f^{-1}(e_s)$.
- 2.1. Let $(v_s, u_s) = e_s$. Let $x = \text{FIND}(v_s)$ and $y = \text{FIND}(u_s)$.
- 2.2. (Search from vertex x and y to root v_r, respectively. Start with greater depth vertex.) If $l(x) > l(y)$, then set $u = z$. Otherwise set $u = y$. Set $z = p(u)$. Assign the edge (u, z) to edge set $f^{-1}(e_s)$. Set $z_r = \text{FIND}(z)$ and process UNION(z_r, u). If $l(x) > l(y)$, then set $z_r = x$. Otherwise, set $z_r = y$.
- 2.3. (Stop when a common ancestor v_k is found.) If $x = y$, then a common ancestor is found and go to Step 2 for next s. Otherwise, go to Step 2.2.

Observe that it takes $O(q)$ number of UNION and $O(q)$ number of FIND in the procedure tree-expansion. The total time complexity in Step 2 is $O(q \cdot \alpha(p, q))$. Thus it takes $O(q \log q)$ time in the modified Algorithm 1.

4. An $O(p^2)$ time algorithm for the 1-MVE problem

In this section, we present an $O(p^2)$ time algorithm to find the most vital edge in a weighted graph. The input form is the weighted matrix $[w(i, j)]_{i,j=1,2,\ldots,p}$. We assume $w(i, j) = \infty$ if there is no edge to connect vertexes i and j, and also assume $w(i, i) = \infty$ for all i.

Algorithm 2

Step 1. Apply Prim's algorithm to find the minimum spanning tree T_G of G. Then, for all edges (i, j) in T_G, set $w(i, j) = \infty$.

Step 2. Pick any vertex v as the root of T_G. Let tree T_i denote the subtree of T_G with root i. Process the following procedure for each vertex i of T_G in postorder.
- 2.1. If vertex i is a leaf node, then set $c(i, j) = w(i, j)$, $j = 1, 2, \ldots, p$.
- 2.2. If vertex i is not a leaf node with child nodes $i_1, i_2, \ldots, i_k$, then set $c(i, j) = \min\{w(i, j), c(i_1, j), c(i_2, j), \ldots, c(i_k, j)\}$, $j = 1, 2, \ldots, p$.
- 2.3. Find $c(i, j^*) = \min\{c(i, j) \mid \text{vertex } j \text{ is not in subtree } T_i\}$ and set $f((i, p(i))) = (j^*, k)$, with $w(j^*, k) = c(i, j^*)$.

Step 3. For each edge e in T_G, compute $w(T_{G-e}) = w(T_G) - w(e) + w(f(e))$. Find the edge e^* such that $w(T_{G-e^*}) = \max\{w(T_{G-e}) \mid \text{the edge } e \text{ is in } T_G\}$.

Obviously, Algorithm 2 correctly computes $f(e)$ for every edge e in T_G. Hence, we can easily compute the most vital edge in G. We also illustrate Algorithm 2 by using the example in Fig. 1. The input matrix is

$$\begin{pmatrix} \infty & 30 & \infty & \infty & \infty & 20 \\ 30 & \infty & 10 & 45 & \infty & \infty \\ \infty & 10 & \infty & 40 & 50 & 25 \\ \infty & 45 & 40 & \infty & 35 & 55 \\ \infty & \infty & 50 & 35 & \infty & 15 \\ 20 & \infty & 25 & 55 & 15 & \infty \end{pmatrix}$$

Step 1 finds the minimum spanning tree T_G shown in Fig. 2 and updates the weighted matrix as follows:

$$\begin{pmatrix} \infty & 30 & \infty & \infty & \infty & \infty \\ 30 & \infty & \infty & 45 & \infty & \infty \\ \infty & \infty & \infty & 40 & 50 & \infty \\ \infty & 45 & 40 & \infty & \infty & 55 \\ \infty & \infty & 50 & \infty & \infty & \infty \\ \infty & \infty & \infty & 55 & \infty & \infty \end{pmatrix}$$

In Step 2, we pick vertex 6 as the root and find the entering edges $f((i,\ p(i)))$, $i = 1, 2, \ldots, 5$. For example, vertex 1 is a leaf node and then

$$(c(1,1),\ c(1,2), \ldots, c(1,6)) = (\infty, 30, \infty, \infty, \infty, \infty).$$

Thus, $c(1,2) = \min\{c(1,1),\ c(1,2), \ldots, c(1,6)\}$ and $f((1,\ p(1))) = f((1,6)) = (1,2)$. Similarly, we find $f((2,3)) = (2,1)$, $f((3,6)) = (2,1)$, $f((4,5)) = (4,3)$, and $f((5,6)) = (4,2)$. Step 3 is the same as that in Algorithm 1.

It is known that it takes $O(p^2)$ and $O(p)$ in Steps 1 and 3, respectively. Let d_i be the degree for vertex i in T_G. Note that it takes $O(p(1+d_i))$ to find $f((i,\ p(i)))$. The total time in Step 2 is

$$O\left(p\sum_{i=1}^{p}(1+d_i)\right) = O(p^2).$$

Hence the time complexity for Algorithm 2 is $O(p^2)$.

References

[1] M.O. Ball, B.L. Golden and R.V. Vohra, Finding the most vital arcs in a network, *Oper. Res. Lett.* **8** (1989) 73–76.

[2] H.W. Corley and D.Y. Sha, Most vital links and nodes in weighted networks, *Oper. Res. Lett.* **1** (1982) 157–160.

[3] R. Gould, *Graph Theory* (Benjamin/Cummings, Menlo Park, CA, 1988).

[4] J.B. Kruskal Jr, On the shortest spanning sub-tree and the travelling salesman problem, *Proc. Amer. Math. Soc.* **7** (1956) 48–50.

[5] R.C. Prim, Shortest connection networks and some generalisations, *Bell System Tech. J.* **36** (1957) 389–401.

[6] R.E. Tarjan, On the efficiency of a good but not linear set merging algorithm, *J. ACM* **22** (1975) 215–225.