ELSEVIER

Parallel Computing 23 (1997) 1889–1907

PARALLEL COMPUTING

Finding the k most vital edges in the minimum spanning tree problem

Weifa Liang [a,*], Xiaojun Shen [b,1]

[a] Department of Computer Science, Australian National University, Canberra, ACT 0200, Australia

[b] Computer Science Telecommunications Program, University of Missouri-Kansas City, Kansas City, MO 64110, USA

Received 25 April 1996; revised 12 February 1997

Abstract

Let $G(V, E)$ be a weighted, undirected, connected simple graph with n vertices and m edges. The k most vital edge problem with respect to a minimum spanning tree is to find a set S ($\subseteq E$) of k edges in G whose removal results in the greatest increase in the weight of the minimum spanning tree in the remaining graph $G(V, E-S)$. Although for arbitrary k, Frederickson et al. have shown that this problem is NP-hard, it is polynomial time solvable when k is fixed. In this paper we introduce a sparse, weighted k-edge connected certificate of a graph which has been found very useful. By using this certificate, we first present a general algorithm for the problem above. Then, for a fixed $k > 1$, we present efficient sequential and parallel algorithms. Our sequential algorithm runs in time $O(n^{k+1})$. Our parallel algorithm runs in time $O(\log n \log\log n)$ using $O(n^{k+1})$ processors on an EREW PRAM. If the minimum spanning tree of G and the sparse, weighted $(k+1)$-edge connected certificate of G are given, the algorithm runs in time $O(\log n)$ using $O(n^{k+1})$ processors on the same model. Particularly, when $k = 1$, we develop parallel algorithms which require $O(\log n \log\log n)$ time using $O(m + n^2/(\log n \ \log\log n))$ processors on an EREW PRAM, and $O(\log n \log\log n)$ time using $O(m)$ processors on a CREW PRAM respectively, and the algorithm on the EREW PRAM is the fastest, compared with previous known algorithms. © 1997 Elsevier Science B.V.

Keywords: Combinatorial algorithms; Minimum spanning trees; Most vital edges; NC algorithms; Network optimization

* Corresponding author. E-mail: wliang@cs.anu.edu.au.

[1] E-mail: xshen@cstp.umkc.edu.

PII S0167-8191(97)00098-7

1. Introduction

Let $G(V, E)$ be an undirected, weighted, connected simple graph with vertex set V and edge set E. Associated with each edge $e \in E$, there is a real valued weight $w(e)$. A *minimum spanning tree* (MST) of G is a spanning tree with minimum total weight. For the sake of convenience, denote by $MST(G)$ the MST of G, and $w(MST(G))$ the total weight of $MST(G)$. The problem of finding an MST in G has been well studied in the past two decades [2,8]. The best sequential algorithm for the MST problem needs $O(m \log \beta(m, n))$ time [8], where $m = |E|$, $n = |V|$ and $\beta(m, n) = min\{i{:}\log^{(i)} n \leq m/n\}$. In particular, when $m \geq n \log^{(i_0)} n$ for some constant i_0, $\beta(m, n)$ is a constant. The best parallel algorithms for the MST problem require $O(\log n \log\log n)$ time using $O(m+n)$ processors on an EREW PRAM [5], and $O(\log^2 n)$ time using $O(n^2/\log^2 n)$ processors on a CREW PRAM [3] for sparse graphs and dense graphs, respectively.

One closely related problem is the k most vital edge problem which can be formally defined as follows. The *k most vital edge problem* with respect to an MST of G (the k most vital edge problem for short) is to find a set $S^* \subseteq E$, $|S^*| = k$, such that $w(MST(G(V, E - S^*)))$ is maximized. This problem has many practical applications including robust network design and distributed computing [7,9–12]. Obviously, $k \leq \lambda \leq \lceil m/n \rceil$, where λ is the edge connectivity of G. Otherwise, after deleting all edges in a minimum cut of G with any other $k - \lambda$ edges, the remaining graph is disconnected, and there is no MST existing in this remaining graph. Therefore, in this paper we assume that G is $(k+1)$-edge connected at least.

For $k = 1$, the problem becomes a special case of finding the single most vital edge with respect to an MST of G, which has been extensively studied in the literature [9–11,15]. Hsu et al. [9] first considered this problem, and gave $O(n^2)$ time and $O(m \log m)$ time sequential algorithms for dense and sparse graphs respectively. Iwano et al. [11] improved Hsu et al.'s results by giving an $O(t_{\mathrm{MST}} + min\{m\ \alpha(m, n), m + n \log n\})$ time algorithm where t_{MST} is the time used to find an MST of G, and α is the inverse Ackermann's function. Hsu et al. [10] later proposed two parallel algorithms for this problem on an EREW PRAM. One of their parallel algorithms requires $O(n^{1+x})$ time and $O(n^{1-x})$ processors with $0 < x < 1$. The other one requires $O(m \log(m/N)/N + n\alpha(m, n) \log(m/n))$ time and $O(N)$ processors where $N \leq (m \log m)/(n\alpha(m, n) \log(m/n))$. All their algorithms need $\Omega(\log^2 n)$ time clearly. Recently Shen [15] presented another parallel algorithm for this problem. His algorithm requires either $O(\log n)$ time and $O(m \log\log\log n/\log n + n)$ processors on a priority CRCW PRAM in which only the processor holding the minimum value is successful when a write conflict happens, or $O(\log n \log\log n)$ time and $O(m + n^2/(\log n \log\log n))$ processors on a CREW PRAM.

For the case $k > 1$, not much work has been done. The only results are due to Lin et al. [12], Frederickson et al. [7] and Shen [16]. Lin et al. have shown that a generalized version of this problem is NP-complete, where each edge in E is assigned a removal cost and the total removal cost B is bounded. But their proof does not imply the NP-completeness of our problem which is a special case of their general version. Frederickson et al. [7] recently proved that the k most vital edge problem is NP-complete, and presented an approximation algorithm for it. Their approximation algorithm requires $O(min\{km \log n + k^2 n \log n,\ km \log^2 n\})$ time, and the solution delivered by their algorithm is $\Omega(1/\log k)$ times optimal. Shen [16] explored this problem by giving an exact algorithm and a randomized, approximation algorithm. His exact algorithm needs $O(n^k m \log \beta(m, n))$ time when k is fixed. Note that, if k is fixed, there always exists a polynomial algorithm for the k most vital edge problem which generates exact solutions.

In this paper, using the notion of sparse, weighted k-edge connected certificates (defined later), we present a general exact algorithm for the k most vital edge problem. When k is fixed, we suggest efficient sequential and parallel algorithms for this problem. Our sequential algorithm runs in $O(n^{k+1})$ time which improves by an $O(m \log \beta(m, n)/n)$ factor in the time bound over Shen's algorithm [16]. Our parallel algorithm runs in $O(\log n \log\log n)$ time using $O(n^{k+1})$ processors on an EREW PRAM. If both the MST of G and the sparse, weighted $(k+1)$-edge connected certificate of G are given, the algorithm runs in time $O(\log n)$ using $O(n^{k+1})$ processors on an EREW PRAM. In particular, for $k = 1$, we develop an NC algorithm on the EREW PRAM which requires $O(\log n \log\log n)$ time and $O(m + n^2/(\log n \log\log n))$ processors. This algorithm outperforms all previous results on this model. We also suggest another NC algorithm on the CREW PRAM which has the same time complexity as Shen's algorithm [15] on the same model, but we use $O(m)$ processors rather than $O(m + n^2/(\log n \log\log n))$ processors by his algorithm.

The parallel computational models involved in this paper are defined as follows. The EREW PRAM is a model in which only exclusive read and exclusive write can be allowed. The CREW PRAM is a model in which concurrent read is allowed but concurrent write is forbidden. The priority CRCW PRAM is a model in which concurrent read is allowed, and when write conflict happens, only the writing processor with the highest priority is successful.

The rest of this paper is organized as follows. In Section 2 we introduce the notion of sparse, weighted k-edge connected certificates, and show how to use it to solve the k most vital edge problem on G. In Section 3 we present two NC algorithms for the single most vital edge problem which run in the EREW PRAM and the CREW PRAM respectively. In Section 4 we develop efficient sequential and parallel algorithms for the k most vital edge problem with fixed k. In Section 5 we make some generalization regarding the k most vital edge problem. A conclusion is given in Section 6.

2. Finding k most vital edges

Without loss of generality, we first assume that the weight assigned to each edge in G is distinct (in Section 5 we will generalize to non-distinct case) and hence the MST or the minimum spanning forest (MSF for short) of graph $G(V, E - S)$ is unique for every $S \subseteq E$.

A naive approach to attack the k most vital edge problem proceeds as follows. First enumerate all different S of k edges from the set E, and compute the weight of the MST of $G(V, E - S)$ for each S. Then find a S_0 of k edges such that $w(MST(G(V, E - S_0))) \geq w(MST(G(V, E - S)))$ for all other $S \neq S_0$. Thus, S_0 is the solution of the problem which contains the k most vital edges. There are $\binom{m}{k}$ different subsets of k edges in E. So the time used for the k most vital edge problem on G is

$$T_{naive} = \binom{m}{k} t_{MST(G')} \tag{1}$$

where $G' = G(V, E - S)$ with $|S| = k$, and $t_{MST(G')}$ is the worst-case time bound to compute the MST of G'.

When k is fixed, $T_{naive} = \binom{m}{k} t_{MST(G')} = O(m^{k+1} \log \beta(m, n))$.

In the following we show that there exists a better algorithm for this problem. The idea behind our algorithm is that we first extend the notion of sparse k-edge connected certificates of an unweighted, undirected graph used in [13,14] to a weighted, undirected graph G, and define the *sparse, weighted k-edge connected certificate* of G. We then show that the k most vital edge problem on G is exactly equivalent to the k most vital edge problem on the sparse, weighted $(k+1)$-edge connected certificate U_{k+1} of G. As a result, instead of using G, we shall use U_{k+1} to find the k most vital edges. Thus, we reduce the size of the searching space from $\binom{m}{k}$ to $\binom{(k+1)(n-1)}{k}$. Particularly when k is fixed, this improvement on the algorithm's performance is significant.

Consider an unweighted, undirected graph G. Let T_1' be a maximal spanning forest of G, and T_i' be a maximal spanning forest of graph $G_i = G - \cup_{j=1}^{i-1} T_j'$ for $i > 1$. Denote by $U_i' = \cup_{j=1}^{i} T_j'$, the union of the maximal spanning forests $T_1', T_2', \ldots, T_i'$. Then the graph U_k' is called the *sparse k-edge connected certificate* of G [13,14], and U_k' has the following property.

Lemma 1. *[13,14] The graph U_k' defined above is l-edge connected if and only if G is l-edge connected, for any integer l with $0 \le l \le k$.*

Notice that Lemma 1 is always held no matter whether G is a simple graph or not. Now we define the sparse, weighted k-edge connected certificate of G.

Definition 1. Let G be a weighted, undirected graph, T_1 be an MSF of G, and T_i be an MSF of $G_i = G - \cup_{j=1}^{i-1} T_j$ for $i > 1$. Denote by $U_i = \cup_{j=1}^{i} T_j$, the union of the MSFs $T_1, T_2, \ldots, T_i$. The graph U_k is called the sparse, weighted k-edge connected certificate of G.

Lemma 2. *The graph U_{k+1} defined above is $(k+1)$-edge connected if and only if G is $(k+1)$-edge connected at least.*

The proof of Lemma 2 is easy and omitted. Actually it is a corollary of Lemma 1. Recall that in the beginning of this paper we already assumed that G is $(k+1)$-edge connected at least. Therefore, from now on, we assume that U_{k+1} is $(k+1)$-edge connected.

Having defined the sparse, weighted $(k+1)$-edge connected certificate U_{k+1} of G, we now show the following lemma which is the key to developing our algorithms later.

Lemma 3. *If $e \in E - S$ is not an edge in U_{k+1}, then e does not belong to the MST of the graph $G(V, E - S)$ for any $S \subseteq E$, where $|S| \le k$.*

Proof. Let $e=(u, v) \in E$ but $e \notin E(U_{k+1})$ where $E(U_{k+1})$ is the edge set of graph U_{k+1}. Because u and v are connected by $e=(u, v)$, u and v must be in the same connected component in G, and in $G_i = G - \cup_{j=1}^{i-1} T_j$ for all i, $1 \le i \le k+1$. Therefore, there is a unique path P_i between u and v in each T_i, $1 \le i \le k+1$. Since $e=(u, v)$ was not chosen in any T_i $(1 \le i \le k+1)$, $w(e)$ must be larger than the weight of any edge in P_i, i.e., $w(e) > w(e')$, where $e' \in \cup_{i=1}^{k+1} E(P_i)$ and $E(P_i)$ denotes the edge set of P_i.

Now we prove that $e=(u, v)$ cannot be contained in the MST of $G(V, E-S)$ for any $S \subset E$, $|S| \le k$. For the purpose of contradiction, suppose $e=(u, v)$ is contained in such an MST. Then, deleting e from this MST will induce a partition of V, $V = V_S \cup (V - V_S)$, where V_S and $V - V_S$ are the vertex sets of subtrees containing u and v respectively. As we argued above, the edges between V_S and $V - V_S$ form a cut which contains at least one edge from each P_i, $1 \le i \le k+1$. Because $|S| \le k$, there must be an edge $e^* \in E(P_j) - S \subseteq E(U_{k+1}) - S \subseteq E - S$, such that $w(e) > w(e^*)$. Replacing e with e^* in the MST will result in a spanning tree in $G(V, E-S)$ which has less weight than that of the MST. A contradiction. □

Remark. Lemma 3 is equivalent to saying that for any S, $S \subseteq E$ and $|S| \le k$, the MST of $G(V, E-S)$ is equal to the MST of U_{k+1}.

Lemma 3 implies an algorithm for our problem. That is, instead of selecting a subset of k edges from the set E of m edges, we only need to select a subset of k edges from the set $E(U_{k+1})$ of at most $(k+1)(n-1)$ edges. Therefore we have

Theorem 1. *The k most vital edge problem on a weighted, connected, undirected simple graph $G=(V,E)$ can be solved in time $O(\binom{kn}{k} kn \log \beta(kn,n))$ for any $k \ge 1$.*

Proof. The discussion is similar to that one in the naive algorithm. The only difference is that we use the sparse, weighted $(k+1)$-edge connected certificate U_{k+1} of G instead of G itself. Let T_{certi} be the time complexity for solving the k most vital edge problem on G. Then $T_{\text{certi}} = \binom{(k+1)(n-1)}{k} t_{\text{MST}(G')}$, where G' is a sparse, weighted, undirected graph induced by deleting k edges from U_{k+1}, and $t_{\text{MST}(G')}$ is the worst-case time used to find the MST of G'. It is well known that the best sequential algorithm for the MST problem on G' requires $t_{\text{MST}(G')} = O(kn \log \beta(kn, n))$ time. The theorem follows. □

In particular, when k is fixed, the time complexity for the k most vital edge problem on G is:

$$T_{\text{certi}} = \binom{(k+1)(n-1)}{k} t_{\text{MST}(G')} = O(n^{k+1} \log \beta(kn, n)). \tag{2}$$

It must be mentioned that Shen [16] also suggested an exact algorithm for the k most vital edge problem on G. His algorithm needs $O(n^k m \log \beta(m, n))$ time when k is fixed, which is obviously inferior to our result above. We later will show how to improve the time bound in Eq. (2) further to $O(n^{k+1})$ by using the technique in Section

4, which improves by an $O(m \log \beta(m, n)/n)$ factor over the time bound of Shen's algorithm.

3. NC algorithms for finding the single most vital edge

In this section we present NC algorithms for the special case $k = 1$ on both the EREW PRAM and the CREW PRAM.

Let T_1 be the MST of G, $E(T_1)$ be the edge set of T_1. Define the replacement edge $r(e)$ of an edge $e \in E(T_1)$ as follows. Delete the edge $e = (u, v)$ from T_1. As a result, the vertex set V is divided into two subsets W and $V - W$ which are the vertex sets of the subtrees including u and v respectively. Let $Q = (W \times (V - W)) \cap E - \{e\}$, and e' be such a non-tree edge that $w(e') = min\{w(e'') : e'' \in Q\}$. If $Q \neq \emptyset$, define $r(e) = e'$, and this $r(e)$ is unique because the weight associated with each edge in E is distinct by our initial assumption. Otherwise, there does not exist a replacement edge for e, and e is a bridge of G. In this case, we know that the remaining graph is disconnected after deleting e from G, therefore no MST exists in this remaining graph. Thus, the edge e is the single most vital edge. For the discussion convenience later, we set $w(r(e)) := +\infty$ if e is a bridge of G.

Now we consider the single most vital edge problem on G. Lin et al. [12] observed that the single most vital edge must be in T_1. Iwano et al. [11] further showed that an edge $e^* \in E(T_1)$ is the single most vital edge if and only if it satisfies the following equation.

$$w(e^*) = \max\{w(r(e)) - w(e) : e \in E(T_1)\} \tag{3}$$

The Eq. (3) implies that the edge $r(e)$ is an edge of the MST in the remaining graph after deletion of e from G. Obviously e^* can be obtained easily by computing $r(e)$ for all $e \in E(T_1)$ in parallel. Thus, the NC algorithm for this problem depends on how to compute $r(e)$ for every $e \in E(T_1)$ efficiently. Let T_2 be the MST (MSF) of the graph $G - T_1$, and $E(T_2)$ be the edge set of T_2. Before we continue, we re-produce the following lemma by Iwano et al. [11].

Lemma 4. *[11] Let T_1 and $E(T_1)$ be defined as above, new(e) be an edge weight function such that new(e) = 0 for $e \in E(T_1)$, and new(e) = $B - w(e)$ for $e \notin E(T_1)$ where $B = max\{w(e) : e \in E\} + \delta$ and δ is a positive constant. Let MaxST(G) be a maximum spanning tree with respect to the above defined weight function new(e), and E(MaxST(G)) be the edge set of MaxST(G). For an edge $e \in E(T_1)$, there is a replacement edge $r(e) \in E(MaxST(G))$; that is, $T_1 - \{e\} \cup \{r(e)\}$ is a minimum spanning tree of $G - \{e\}$.*

Lemma 4 suggests an algorithm for computing all $r(e)$ from the tree MaxST(G). Let w(MaxST(G)) denote the weight of MaxST(G) and $n' = |E(\text{MaxST}(G))|$. Then

$$w(\text{MaxST}(G)) = n'B - \sum_{i=1}^{n'} w(e_i), \text{ where } e_i \in E(\text{MaxST}(G)). \tag{4}$$

Notice that, for any given undirected, weighted simple graph G'' with n'' vertices no matter whether it is connected or not, the number of edges n' in all spanning trees (or

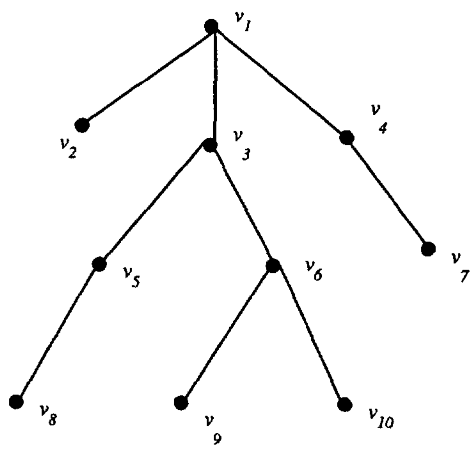

Fig. 1. A tree T_1.

spanning forests) of G'' is the same. This means that the value $n'B$ in Eq. (4) is fixed when G is given. By Eq. (4), we know that, in order to make $w(\text{MaxST}(G))$ maximized, we only need to find a spanning tree (or a spanning forest) T' in $G - T_1$ such that $\sum_{i=1}^{n'} w(e'_i)$ is minimized where $e'_i \in E(T')$. By the definition of the MST (MSF), T' is the MST (MSF) of $G - T_1$, i.e, $T' = T_2$. Obviously $\text{MaxST}(G) = T' = T_2$. Shen [15] noticed this and presented it explicitly by the following lemma. Note that the edges in T_1 are not included in MaxST(G) because the new weights associated with the edges in T_1 are zeros, whereas all the new weights associated with the edges in *MaxST*(G) are positive by Lemma 4.

Lemma 5. *[15] For any edge e in T_1, $r(e)$ is an edge in the MST of the graph $G - T_1$.*

Now we present an approach for computing $r(e)$ for each $e \in E(T_1)$. Assume that T_1 is a rooted tree. We do the pre-order transversal on T_1 and assign to each vertex v the pre-order numbering pre(v) and the number of descendents nd(v) (including v itself). The following example (see Fig. 1) shows how to compute pre()s and nd()s on a tree T_1 consisting of 10 vertices $v_1, v_2, \ldots, v_{10}$ where v_1 is the root of T_1. Then

$\text{pre}(v_1) = 1$,	$\text{nd}(v_1) = 10$;
$\text{pre}(v_2) = 2$,	$\text{nd}(v_2) = 1$;
$\text{pre}(v_3) = 3$,	$\text{nd}(v_3) = 6$;
$\text{pre}(v_4) = 9$,	$\text{nd}(v_4) = 2$;
$\text{pre}(v_5) = 4$,	$\text{nd}(v_5) = 2$;
$\text{pre}(v_6) = 6$,	$\text{nd}(v_6) = 3$;
$\text{pre}(v_7) = 10$,	$\text{nd}(v_7) = 1$;
$\text{pre}(v_8) = 5$,	$\text{nd}(v_8) = 1$;
$\text{pre}(v_9) = 7$,	$\text{nd}(v_9) = 1$;
$\text{pre}(v_{10}) = 8$,	$\text{nd}(v_{10}) = 1$.

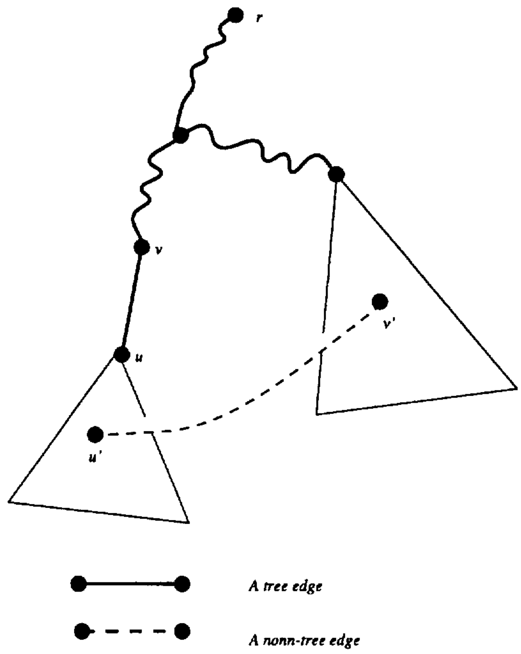

Fig. 2. An illustration of Eq. (5).

Having done the assignment above, we now consider how to compute $r(e)$ for every edge $e=(u, v) \in E(T_1)$. Suppose that v is the parent of u in T_1. By Lemma 5, for any edge $e \in E(T_1)$ we have the following formula.

$$\begin{aligned} w(r(e)) = \min\{w(e''): e'' = (u', v') \in E(T_2), \text{pre}(u) \le \text{pre}(u') \\ < \text{pre}(u) + \text{nd}(u), \text{ and either pre}(v') \\ < \text{pre}(u) \text{ or pre}(v') \ge \text{pre}(u) + \text{nd}(u)\} \end{aligned} \quad (5)$$

Eq. (5) says that, if a non-tree edge $e''=(u'', v'')$ is the replacement edge of a tree edge $e=(u, v)$, it must satisfy all the following conditions: (i) $e'' \in E(T_2)$; (ii) one endpoint u' of e'' must be in the subtree T_v and the other endpoint v' must not be in T_v; and (iii) e'' is the edge with the minimum weight among all those edges which satisfy (i) and (ii). Fig. 2 gives such an illustration.

In the following we first present an NC algorithm for this problem on an EREW PRAM. In order to compute $r(e)$ for each $e \in E(T_1)$, we need a copy of T_2, and a copy of both pre(v) and nd(v) for every $v \in V$ because we use the EREW PRAM. So, there are a total of $n-1$ copies of T_2, pre(v) and nd(v) needed. This can be implemented in $O(\log n)$ time using $O(n^2/\log n)$ processors on this model by the broadcasting technique. It can also be implemented in $O(\log n \log\log n)$ time using $O(n^2/(\log n \log\log n))$ processors on this model by Brent's Theorem. Now for each edge $e \in E(T_1)$, there is a corresponding copy of T_2 as well as a copy of pre(v) and

$\text{nd}(v)$ for each $v \in V$. By Eq. (5) above, $r(e)$ can be obtained in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM because of $|E(T_2)| \leq n-1$. Note that $r(e)$ can also be obtained in $O(\log n \log\log n)$ time using $O(n/(\log n \log\log n))$ processors on the same model by Brent's Theorem. So, the total number of processors used for computing all replacement edges is $O(n^2/(\log n \log\log n))$ if all $r(e)$ need to be found in $O(\log n \log\log n)$ time. By the discussions above, we have the following theorem.

Theorem 2. *Given a weighted, connected, undirected simple graph $G(V,E)$, the single most vital edge can be found in $O(\log n \log\log n)$ time using $O(m + n^2/(\log n \log\log n))$ processors on an EREW PRAM.*

Proof. Finding the trees T_1 and T_2 require $O(\log n \log\log n)$ time using $O(m)$ processors on an EREW PRAM respectively, assuming $m \geq n-1$. It is well known that the assignment of the pre-order numbering $\text{pre}(v)$ and the number of descendents $\text{nd}(v)$ of v in a rooted tree can be done in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM by Euler transversal and the tree contraction technique [1]. Given T_2, $\text{pre}(v)$ and $\text{nd}(v)$ for each vertex v in T_1, the computation of all replacement edges in T_1 can be done in $O(\log n \log\log n)$ time using total $O(n^2/(\log n \log\log n))$ processors on an EREW PRAM. Then, Eq. (1) can be computed in $O(\log n)$ time with $O(n/\log n)$ processors. Therefore, the theorem follows. □

Next we consider how to compute $r(e)$ for each $e \in E(T_1)$ on a CREW PRAM. In this model we do not need to make $n-1$ copies of T_2 initially. We proceed by the following lemma.

Lemma 6. *Let T_1 be the MST of G, and $E(T_1)$ be the edge set of T_1. Then all $r(e)$ can be obtained in $O(\alpha(2n, n)\log n)$ time using a total of $O(n/\log n)$ processors on a CREW PRAM, where $e \in E(T_1)$ and α is the inverse of the Ackermann's function.*

Proof. Dixon et al. [6] have studied the minimum spanning tree sensitivity analysis problem on a graph with n vertices and m edges, and presented an NC algorithm for this problem. Their algorithm requires $O(\alpha(m, n)\log n)$ time and $O((m+n)/\log n)$ processors on a CREW PRAM. One important part in their algorithm is to compute $r(e)$ for all tree edges $e \in E(T_1)$ which was denoted by $b(e)$ in their paper. We will only need that part of their algorithm. By Lemmas 3 and 5, we know that $\{r(e): e \in E(T_1)\} \subseteq E(T_2)$ and $T_2 \subset U_2$. Therefore we can work on the graph U_2 – the sparse, weighted 2-edge connected certificate of G instead of G to find all $r(e)$. Note that $|E(U_2)| \leq 2n - 2$. So, by applying Dixon et al.'s algorithm to the graph U_2, we can find all $r(e)$ in $O(\alpha(2n, n)\log n)$ time using a total of $O(n/\log n)$ processors on a CREW PRAM, where $e \in E(T_1)$. □

Based on Lemma 6, we now state the following theorem.

Theorem 3. *Given a weighted, connected, undirected simple graph $G(V, E)$, the single most vital edge can be found in $O(\log n \log\log n)$ time using $O(m)$ processors on a CREW PRAM.*

Proof. The construction of U_2 requires $O(\log n \log\log n)$ time and $O(m+n)$ processors on an EREW PRAM using the fastest MST algorithm on this model. So, U_2 also can be found in the same time and processor bounds on a CREW PRAM because the EREW PRAM is more restricted, compared with the CREW PRAM. The computation of all $r(e)$ can be done in $O(\alpha(2n, n)\log n)$ time using $O(n/\log n)$ processors on a CREW PRAM by Lemma 6, where $e \in E(T)$. Notice that $\alpha(2n, n) \leq c \log\log n$, where c is a constant. The Eq. (1) can be computed in $O(\log n)$ time using $O(n/\log n)$ processors on a CREW PRAM. Since G is connected, $m > n-1$. So, the theorem follows. □

The time complexity of our algorithm on the EREW PRAM is a substantial improvement on that in Hsu et al.'s algorithm which needs $\Omega(\log^2 n)$ time on the same model. The cost to achieve our time bound is by employing more processors and incorporating currently the most efficient MST algorithm [5]. However, the number of processors in our algorithm is no more than $O(n^2)$. With the same time-processor product, our algorithm is much faster. The time complexity of our algorithm in the CREW PRAM is the same as Shen's [15] on this model, but we use $O(m)$ processors rather than $O(m + n^2/(\log n \log\log n))$ processors in his algorithm. Particularly, when we consider sparse graphs, our algorithm is much better than Shen's algorithm. For example, consider a sparse graph with n vertices and $m = O(n)$ edges, our algorithm requires $O(\log n \log\log n)$ time and $O(n)$ processors on this model, but his algorithm needs $O(n^2/(\log n \log\log n))$ processors with the same time complexity.

4. Efficient sequential and parallel algorithms with fixed $k \geq 2$

In this section for the case of fixed $k \geq 2$, we provide efficient sequential and parallel algorithms for the k most vital edge problem.

By Theorem 1, we can easily suggest a parallel algorithm for the k most vital edge problem with arbitrary k. The algorithm can be described as follows. First, find the sparse, weighted $(k+1)$-edge connected certificate U_{k+1} of G. This can be done in $O(k \log n \log\log n)$ using $O(m+n)$ processors on an EREW PRAM by applying the best parallel algorithm for the MST problem [5]. Then select a set S of k edges from $E(U_{k+1})$ arbitrarily, and delete all edges of S from U_{k+1}. Third, compute the MST of the remaining graph $U_{k+1} - S$. This can be done in $O(\log n \log\log n)$ time using $O(kn)$ processors on an EREW PRAM. There are a total of $\binom{(k+1)(n-1)}{k}$ different S sets, thus, the total number of processors used in this step is $(kn\binom{(k+1)(n-1)}{k})$. Finally, select one set S^* from these $\binom{(k+1)(n-1)}{k}$ sets such that $w(MST(U_{k+1} - S^*))$ is maximized. The set S^* can be found in $O(\log\binom{(k+1)(n-1)}{k})$ time using $\binom{(k+1)(n-1)}{k}$ processors on an EREW PRAM. Therefore, there exists a simple parallel algorithm for this problem which requires $O(k \log n \log\log n)$ time and $O(kn\binom{kn}{k})$ processors on an EREW PRAM.

When k is fixed, we have the following corollary.

Corollary 1. *The k most vital edge problem on $G(V, E)$ with fixed k can be solved in $O(\log n \log\log n)$ time using $O(n^{k+1})$ processors on an EREW PRAM.*

In the following we will show how to obtain a better algorithm for this problem. We first develop a simple sequential algorithm. Then we present an efficient sequential implementation of this algorithm. Finally we show how to parallelize this sequential algorithm in Section 4.2.

4.1. A sequential algorithm and its efficient implementation

As preparation, we present some lemmas related to the k most vital edge problem.

Lemma 7. *Let T be the MST of G, and $E(T)$ be the edge set of T. Assume that S^* is the set of k edges whose deletion results in the maximum increase of in the weight of the MST of $G(V, E-S^*)$. Then $E(T) \cap S^* \neq \emptyset$.*

Proof. The proof of this lemma is easy. The approach we adopted is similar to that one used in [9]. If $E(T) \cap S^* = \emptyset$, then $G(V, E-S^*)$ contains the tree T. Thus the MST of $G(V, E-S^*)$ is T. However, replacing any edge $e \in E(T)$ with $r(e)$ will result in a spanning tree of larger weight. A contradiction. □

Recall that T is the MST of G, and S^* is the set of k edges whose deletion results in the maximum increase of in the weight of the MST of $G(V, E-S^*)$. By Lemma 7, $E(T) \cap S^* \neq \emptyset$. Let $|E(T) \cap S^*| = r$. In the following we construct an auxiliary undirected, weighted, multigraph $G_0 = (V_0, E_0)$. (A graph is a multigraph if, for a pair of vertices u and v, there exist multiple edges between them.) We later show that the k most vital edge problem on G can be reduced to the $(k-r)$ most vital edge problem on G_0 for a given r.

The construction of G_0 is as follows. Delete the r edges in $E(T) \cap S^*$ from T. After that, T becomes a forest F of $r+1$ trees, $r \leq k$. For each tree in F, we use its root label to label all vertices in it. Now the vertex set V_0 of G_0 consists of all trees in F. Let T_u and T_v be two trees in F, an edge $(T_u, T_v) \in E_0$ with weight $w(e)$ if and only if there exists an edge $e = (u, v) \in E$ such that $u \in T_u$ and $v \in T_v$ and $T_u \neq T_v$. Obviously G_0 is a multigraph with $r+1$ vertices and $|E_0| \leq |E - E(T)| = m - (n-1) < m$ edges.

From the graph G_0, we construct a subgraph $G_1(V_1, E_1)$ of G_0 as follows. Make $V_1 = V_0$. Let T_u and T_v be two vertices in G_0, sort the edges in E_0 which connect T_u and T_v in increasing order, using the weight associated with each edge as the key. The first $k-r+1$ edges are included in E_1. Note that the number of edges in G_1 is $|E_1| = O(r^2(k-r+1)) = O(k^3)$ which is independent of n and m.

Lemma 8. *Let G_0 and G_1 be defined as above. Then the k' most vital edge problem on G_0 is equivalent to the k' most vital edge problem on G_1, where $0 \leq k' \leq k-r$.*

Proof. The approach we adopted is to show that the sparse, weighted $(k'+1)$-edge connected certificates of both G_0 and G_1 are the same. Then from Lemma 3, this lemma follows. So, in the rest of this proof we show that the sparse, weighted $(k'+1)$-edge connected certificates of both G_0 and G_1 are the same. Notice that G_1 is a subgraph of G_0.

Let T_0^1 be the MSF of G_0, and T_0^i be the MSF of $G_0^i = G_0 - \cup_{j=1}^{i-1} T_0^j$. By the definition in Section 2, the graph $G'' = \cup_{i=1}^{k'+1} T_0^i$ is the sparse, weighted $(k'+1)$-edge connected certificate of G_0. We observe that, for each pair of vertices T_u and T_v in G'', there are at most $k'+1$ edges between them, and each of these $k'+1$ edges belongs to a unique forest T_0^i only, $1 \le i \le k'+1$. Obviously these edges between T_u and T_v are contained in the first $k'+1$ minimum weighted edges. By the definition of G_1, these $k'+1$ edges are included in G_1. Therefore, G'' is a subgraph of G_1. Thus, the sparse, weighted $(k'+1)$-edge connected certificate of G_1 is also G''. The lemma follows. □

Lemma 9. *Given T, S^*, F and G_0 as defined above, and $|E(T) \cap S^*| = r \ne 0$, the MST in $G(V, E - S^*)$ is equal to the union of the MST in $G_l(V_l, E_l - S^*)$ and F.*

Proof. Let T^* be the MST of $G(V, E - S^*)$. First we show that any edge e of F must be contained in the MST of $G(V, E - S^*)$. Suppose e is not, then adding e to T^* will form a cycle, and $w(e)$ must be larger than any other edges' weight on the cycle, because otherwise we can obtain a smaller weighted spanning tree by replacing an edge e' with e if $w(e') > w(e)$. However, if $w(e)$ is larger than any other edges' weight on the cycle, then e should not have been included in T — the MST of G which leads to a contradiction. So, any edge in F must be contained in the MST of $G(V, E - S^*)$.

Second, it is clear that any edge (not in F) connecting two vertices of a tree in F cannot be included in T^*. Therefore, $E(T^*) - E(F) \subseteq E(G_0)$. Let G^* be the subgraph of G_0 induced by $E(T^*) - E(F)$. Obviously G^* must be connected, otherwise the trees in F will not be connected by the edges in $E(T^*) - E(F)$. Also, G^* must be acyclic, otherwise, a cycle in T^* would occur. Therefore, G^* must be a spanning tree of G_0. On the other hand, any spanning tree of G_0 together with F will form a spanning tree of $G(V, E - S^*)$. Since $w(F)$ is fixed, $T^* = F \cup MST(G_0)$, and $w(T^*) = w(F) + w(MST(G_0))$. Since $MST(G_0) = MST(G_1)$, the lemma follows. □

By Lemmas 7, 8 and 9, we have the following recursive, sequential algorithm for the k most vital edge problem. The input of the algorithm is a weighted, undirected graph G (G may be a multigraph), n and k. The output is the set S of k edges, and the maximized weight W of the MST of the graph $G(V, E - S)$.

```
Procedure Find_Vital_Edges(G, n, k, W, S);
  MAX := −∞; S* := ∅;
  compute T, the MST of G;
  if k = 0 then
        W := w(MST(G)); S := ∅; exit;
  else
        for r := 1 to k do
            for each set S0 of r edges do
            /* S0 is selected from E(T) systematically, one by one. */
                    delete the edges in S0 from T;
                    construct the graph G1;
                    Call Find_Vital_Edges(G1, r + 1, k − r, W1, S1);
                    if w(T) − w(S0) + W1 > MAX
                          then S* := S0 ∪ S1;
```

```
                    MAX := w(T) − w(S_0) + W_1
            endif
        endfor
    endfor;
    S := S*; W := MAX;
endif
```

Now we analyze the time complexity of this algorithm.

Lemma 10. *The k most vital edge problem on G with fixed k can be solved in time $O(n^k m \log n)$.*

Proof. The time complexity of the algorithm above can be expressed by the following recursive equation.

$$T_A(n, m, k) = \sum_{r=1}^{k} \binom{n-1}{r} \Bigg[t_{\mathrm{MST}(G)}(m, n) + T_A\left(r+1, \frac{r(r+1)}{2}(k-r+1), k-r\right) + T_{G_1}(m, n) \Bigg] \quad (6)$$

where

· $T_A(n, m, k)$ is the time complexity of the algorithm above for the k most vital edge problem on a graph (either a simple graph or a multigraph) with n vertices and m edges;

· $t_{\mathrm{MST}(G)}(m, n)$ is the time complexity for constructing the MST of the graph G with n vertices and m edges;

· $TG_1(m, n)$ is the time complexity for constructing the graph G_1 starting from a graph with n vertices and m edges.Obviously $t_{\mathrm{MST}(G)}(m, n) = O(m \log \beta(m, n))$, $TG_1(m, n) = O(m \log n)$ because the construction of G_1 needs sorting all edges in G_0 which costs $O(m \log n)$ time, and by the naive algorithm in Section 2, $T_A(r+1, r(r+1)/2(k-r+1), k-r) \le C$ where C is a constant. The reason is that G_1 contains constant vertices and edges if k is fixed. Thus

$$\begin{aligned} T_A(n, m, k) &= \sum_{r=1}^{k} \binom{n-1}{r} \Bigg[t_{MST(G)}(m, n) + T_A\left(r+1, \frac{r(r+1)}{2}(k-r+1), k-r\right) + T_{G_1}(m, n) \Bigg] \\ &\le (n + n^2 + \ldots + n^k)(m \log \beta(m, n) + C + m \log n) \\ &= O(n^k m \log n) \text{ with fixed } k. \quad \square \end{aligned}$$

The algorithm above can be further improved by applying Lemma 3, i.e, we use the sparse, weighted $(k+1)$-edge connected certificate U_{k+1} of G instead of G itself as the initial input. Meanwhile, we use the sparse, weighted $(k-r+1)$-edge connected certificate G'' of G_1 instead of G_1. As a result, we derive the time complexity of the algorithm above is $O(n^{k+1} \log n)$ with fixed k. Though the time complexity of the algorithm in Lemma 10 is inferior to that in Eq. (2), in the following we shall present an efficient implementation of this algorithm to improve its time complexity. As a result, the algorithm can run in $O(n^{k+1})$ time when k is fixed. This result improves the results above as well as the result in [16]. Furthermore, this algorithm also leads to a better parallel algorithm for the k most vital edge problem in Section 4.2.

Now we present an efficient sequential implementation of the algorithm above. Recall that if we already knew the specified r most vital edges in T, by the arguments above, the other $k-r$ most vital edges can be identified from the multigraph G_1. Since G_1 contains constant vertices and edges, by the naive algorithm in Section 2, we can easily identify the $k-r$ most vital edges in G_1 in $O(1)$ time. So how to construct the graph G_1 is the key. At this before, the construction of G_1 uses the sorting routine which costs $O(|E(U_{k+1})| \log |E(U_{k+1})|) = O(n \log n)$ time. Here we show that there exists a better way to construct G_1, and this construction can be done in $O(n)$ time. We proceed as follows. Let the r most vital edges in T be $e_1, e_2, \ldots, e_r$. First, delete these r edges from T. Then compute the connected component in $T - \{e_1, e_2, \ldots, e_r\}$, and label all vertices in a connected component with a unique identification, which can be implemented in $O(n)$ time because this graph is a forest. Finally, let us consider the graph G_1 which contains $r+1$ vertices, each one corresponds to a connected component. There are $O((r+1)^2) = O(k^2)$ vertex pairs in G_1. For each pair of vertices in G_1, for example C_1 and C_2, we compute the edges between them by the following procedure. Let R be all edges in U_{k+1}. We delete all other edges whose endpoints are not labeled by C_1 and C_2 respectively. Let the remaining edge set be R'. Obviously R' can be obtained in $O(n)$ time because $|R'| \le |R| = O(n)$. Now select the $(k-r)+1$ smallest element from R' using the weight associated with each edge as the searching key. Assuming e is the $(k-r)+1$ smallest element of R'. Compare all other elements e' in R' with e, if $w(e') \le w(e)$, e' is kept in R'; otherwise e' is deleted from R'. Let R'' be a subset of R' which contains all remaining elements in R'. Then, by the definition, R'' is the edge set between vertices C_1 and C_2 in G_1. R'' can be obtained in $O(n)$ time because selecting an element from a set of $O(n)$ elements costs $O(n)$ time [2]. Therefore, the construction of G_1 requires $O(k^2 n) = O(n)$ time because G_1 has $O(k^2)$ vertex pairs. Having the graph G_1, the $k-r$ most vital edges in G_1 can be identified in constant time, and the weight, denote by $w(G_1, k-r)$, of the MST in the remaining graph of G_1 by deleting these $k-r$ edges from G_1 can be obtained in constant time. Therefore, the weight of the remaining graph induced on G by deleting these k most vital edges from G can be derived, i.e, the weight is $w(\mathrm{MST}(G)) - \sum_{i=1}^{r} w(e_i) + w(G_1, k-r)$ which can be obtained in constant time. Now we state the following theorem.

Theorem 4. *The k most vital edge problem on a weighted, connected, undirected simple graph $G=(V, E)$ can be solved in time $O(n^{k+1})$ with fixed $k \ge 1$.*

Proof. The MST of G can be obtained in time $O(m \log \beta(m, n)) = O(n^2)$ by the algorithm in [8]. There are $\binom{n-1}{r} = O(n^r)$ ways to chose r edges from T. For each specified r edges, if each of them is among the k most vital edges in G, the other $k - r$ most vital edges in G can be identified in $O(n)$ time by G_1 because the construction of G_1 needs $O(n)$ time. The weight of the MST in the remaining graph after deleting these $r + (k - r)$ edges from G can be obtained in $O(n)$ time because of the same reason. Let S be such a set of the potential k most vital edges. Since $1 \le r \le k$, we need to chose one S_0 of k edges from a total $\sum_{r=1}^{k} \binom{n-1}{r} = O(n^k)$ sets of S such that the MST in the remaining graph $G(V, E - S_0)$ has the maximum weight. By the discussion above, for each of these sets of S, it needs $O(n)$ time to compute the weight of the MST in the remaining graph $G(V, E - S)$. So, for the k most vital edge problem, we need $O(n^{k+1})$ time to solve it. □

4.2. A parallel implementation

Assume that the edges in T have been numbered from 1 to $n - 1$. The parallelization of the algorithm above is not difficult. We make $O(n^k)$ copies of T, the MST of G. This can be implemented by applying a broadcasting technique in which a single value is broadcasted into $O(n^k)$ places using prefix computation. Thus, this step needs $O(k \log n) = O(\log n)$ time and $O(n^{k+1})$ processors on an EREW PRAM because there are $n - 1$ edges in T. Then for different r with $1 \le r \le k$, there are a total of $\binom{n-1}{r}$ subsets of r edges in T, and each such a subset is numbered. For a specific subset S numbered i, there exists a corresponding copy of T, we delete these r edges in S from this copy of T. As a result, we obtain a spanning forest F. For every vertex in F, compute its connected component identification. Here we label each vertex with the root's label of the tree to which it belongs. This can be implemented by applying the tree contraction algorithm of Abrahamson et al. [1] to F which requires $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM. Having done the above, it is easy to construct $G_0(V_0, E_0)$. Obviously $|E_0| \le |E| - (n - 1) - |S| < m$. Now we give the details to construct the graph G_1. Suppose that $V = \{1, 2 \dots, n\}$, and every undirected edge (u, v) in E_0 is stored in the form $(min\{u, v\}, max\{u, v\})$. We label each endpoint of the edges in E_0 by its connected component identification. Sort all edges in E_0 in increasing order, using the labels of the endpoints of each edge as the primary key, and the weight associated with this edge as the second key. Let the sorted sequence be R. Define a subsequence R' of R as an interval if the endpoints of all edges in R' have the same labels. Delete all other edges in R' except the first $k - r + 1$ edges. This can be implemented by compressing R using prefix computation. The remaining edges in the compressed R is the edges of G_1. Sorting and prefix computation can be done in $O(\log n)$ time using $O(m)$ processors on an EREW PRAM [4] because there are $O(m)$ edges to be sorted. Note that G_1 is a multigraph with $k + 1$ vertices and ck^3 edges at most, and $ck^3 \le m$, where c is a constant. Finally, find the sparse, weighted $(k - r + 1)$-edge connected certificate G'' of G_1 which requires $O(k \log k \log\log k)$ time using $O(k^3)$ processors on an EREW PRAM because G_1 contains $O(k^3)$ edges at most. It is easy to derive that the depth of recursion of the algorithm is bounded by $O(k)$. When k is fixed, we have

Table 1
The comparison of the results

Authors	Sequential	EREW	CREW	CRCW	k
Hsu et al. [9]	$O(n^2)$ or $O(m \log n)$				$k = 1$
Hsu et al. [10]		$T = O(n^{1+x})$, $P = O(n^{1-x})$, $0 \leq x \leq 1$			$k = 1$
Iwano et al. [11]	$O(t_{MST} + min\{m\alpha(m, n), m + n \log n\})$				
Liang et al. this paper		$T = O(\log n \log\log n)$, $P = O(m + n^2/\log n \log\log n)$	$P = O(m)$	$T = O(\log n \log\log n)$,	$k = 1$
Liang et al. this paper	$O(n^{k+1})$	$T = O(\log n \log\log n)$, $P = O(n^{k+1})$			k is fixed
Shen [15]			$T = O(\log n \log\log n)$, $P = O(m + n^2/(\log n \log\log n))$	$T = O(\log n)$, $P = O(m \log\log\log n/\log n + n)$	$k = 1$
Shen [16]	$O(n^k m \log \beta(m, n))$				k is fixed.

Lemma 11. *Given the MST of G, the k most vital edge problem on G with fixed k can be solved in O(log n) time using $O(mn^k)$ processors on an EREW PRAM.*

Proof. For a fixed r, there are a total of $\binom{n-1}{r}$ different ways to select r edges from T. For a given specific r edges in T, constructing G_i requires $O(\log n)$ time and $O(m)$ processors on an EREW PRAM, $i = 0, 1$. The $k - r$ most vital edge problem on G_1 can be solved in constant time because G_1 contains $O(k)$ vertices and $O(k^3)$ edges at most and k is fixed. Since r is bounded by $1 \le r \le k$, there are a total of $mP(n, r) = m \sum_{r=1}^{k} \binom{n-1}{r} = O(mn^k)$ processors required. The lemma follows. □

One direct application of Lemma 11 is to find the k most vital edges in a graph G with bounded degree d. For this special case, the number of edges of G is $m = O(n)$. Thus, we have

Lemma 12. *Given a weighted, undirected, simple graph G(V,E) with bounded degree d and the MST of G, the k most vital edge problem on G can be solved in O(log n) time using $O(n^{k+1})$ processors on an EREW PRAM.*

Combining Lemmas 11 and 12, we have the following theorem immediately.

Theorem 5. *Given a weighted, undirected, connected simple graph G(V, E), the MST of G, and the sparse, weighted (k + 1)-edge connected certificate of G, the k most vital edge problem on G with fixed k can be solved in O(log n) time using $O(n^{k+1})$ processors on an EREW PRAM.*

Proof. We use U_{k+1} instead of G as the initial input of the algorithm. Note that $m = |E(U_{k+1})| \le (k+1)(n-1) = O(n)$, since k is fixed. By Lemma 11, the theorem follows. □

Note that Theorem 5 holds only when T and U_{k+1} are given. If not so, the preprocessing for the construction of U_{k+1} requires $O(\log n \log\log n)$ time using $O(m+n)$ processors on an EREW PRAM.

We now summarize all known results for solving the k most vital edge problem by given Table 1.

5. Generalization

Until now all our previous discussions are based on the following two assumptions: (1) all the weights associated with the edges in E are distinct; and (2) $G(V, E)$ is $(k+1)$-edge connected at least. In this section we make some generalization regarding the k most vital edge problem without these assumptions.

Firstly, consider the case in which the weights associated with the edges in E are not distinct. This means that the MST of G may not be unique. In this case, to make our previous algorithms work, it needs some rules to break the tie. Here we introduce such a rule, described as follows. Let $V = \{1, 2, \dots, n\}$. For arbitrary two edges $e_1 = (u, v) \in E$ and $e_2 = (x, y) \in E$, we say that the 'weight' of e_1 is smaller than that of e_2 if and only if one of following conditions holds: (i) $w(e_1) < w(e_2)$; (ii) $w(e_1) = w(e_2)$ but $\min\{u, v\} < min\ \{x, y\}$; (iii) $w(e_1) = w(e_2)$ and $\min\{u, v\} = \min\ \{x, y\}$ but $\max\{u, v\} < \max\{x, y\}$. Then for each edge e we use the 'weight' defined above rather than the original $w(e)$ to all our algorithms.

Secondly, consider that G is not $(k+1)$-edge connected but k'-edge connected, $1 \le k' \le k$. By Lemma 1, it is easy to derive that U_{k+1} is k'-edge connected. Therefore there exists a minimum cut S of k' edges in U_{k+1}. When deleting all edges in S from U_{k+1}, the remaining graph is disconnected, therefore no MST exists in this remaining graph, i.e, the weight of the MST in the remaining graph is $+\infty$. By this discussion, we know that, any set S of k edges in U_{k+1} with $S \subset S'$ is a solution of the problem, and our algorithms also work for this case.

6. Conclusions

In this paper we have developed a general exact algorithm for the k most vital edge problem by using the sparse, weighted $(k+1)$-edge connected certificate of a weighted undirected graph. When k is fixed, we present efficient sequential and parallel algorithms for this special case. Because the k most vital edge problem is NP-complete, developing efficient sequential and parallel approximation algorithms with better performance ratios for this problem is our future work.

Acknowledgements

We appreciate the anonymous referees for their helpful comments and suggestions which improve the quality of this paper. Meanwhile we would like to thank Hong Shen for his valuable comments in the initial draft. We also thank Stephanie Griffin for her help in improving the presentation of the paper. The work by the second author was partially supported by UMRB Grant K-3-40627.

References

[1] K. Abrahamson, N. Dadoun, D.G. Kirkpatrick, T. Przytycka, A simple parallel tree contraction algorithm, J. Algorithms 10 (1989) 287–302.
[2] A. Aho, J. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.
[3] F.Y. Chin, J. Lam, I.-N. Chen, Efficient parallel algorithms for some graph problems, Commun. ACM. 25 (1982) 659–665.
[4] R. Cole, Parallel merge sort, SIAM J. Comput. 17 (1988) 770–785.
[5] K.W. Chong, T.W. Lam, Finding minimum spanning trees on the EREW PRAM, Manuscript, Feb., 1996.
[6] B. Dixon, R.E. Tarjan, Optimal parallel verification of minimum spanning trees in logarithmic time, Lect. Notes Comput. Sci. 805 (1994) 13–22.

[7] G.N. Frederickson, R. Solis-Oba, Increasing the weight of minimum spanning trees, Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, January, 1996, pp. 539–546.
[8] H.N. Gabow, Z. Galil, T. Spencer, R.E. Tarjan, Efficient algorithms for finding minimum spanning trees in undirected and directed graphs, Combinatorica 6 (2) (1986) 109–122.
[9] L. Hsu, R. Jan, Y. Lee, C. Hung, Finding the most vital edge with respect to minimum spanning tree in a weighted graph, Inform. Proc. Lett. 39 (1991) 277–281.
[10] L. Hsu, P. Wang, C. Wu, Parallel algorithms for finding the most vital edge with respect to minimum spanning tree, Parallel Comput. 18 (1992) 1143–1155.
[11] K. Iwano, N. Katoh, Efficient algorithms for finding the most vital edge of a minimum spanning tree, Inform. Proc. Lett. 48 (1993) 211–213.
[12] K. Lin, M. Chern, The most vital edges in the minimum spanning tree problem, Inform. Proc. Lett. 45 (1993) 25–31.
[13] H. Nagamochi, T. Ibaraki, A linear time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph, Algorithmica 7 (1992) 583–596.
[14] H. Nagamochi, T. Ibaraki, Computing edge-connectivity in multigraphs and capacitated graphs, SIAM J. Discrete Math. 5 (1992) 54–66.
[15] H. Shen, Improved parallel algorithms for the most vital edge of a graph with respect to minimum spanning tree, Technical Report, Dept. of Computer Sci., Abo Akademi Univ. Finland, 1995. URL: http://www.abo.fi/~mats/cs/publications/reports.
[16] H. Shen, Finding the k most vital edges with respect to minimum spanning tree, Technical Report, Dept. of Computer Sci., Abo Akademi Univ. Finland, 1995. URL: http://www.abo.fi/~mats/cs/publications/reports.