Finding the k Most Vital Edges with respect to Minimum Spanning Trees for $k = 2$ and 3

Weifa Liang[1,2] George Havas[1]

[1] Department of Computer Science and Electrical Engineering
[2] Department of Mathematics
The University of Queensland
Queensland 4072, Australia
{wliang, havas@it.uq.edu.au}

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 minimum spanning trees is to find a set S^* of k edges from E such that the removal of all edges in S^* results in the greatest increase in the weight of the minimum spanning tree in the remaining graph $G(V, E - S^*)$. In this paper we present a better algorithm to solve this problem for $k = 2$ and 3. The proposed algorithms run in times $O(n^2\alpha(3n, n))$ for $k = 2$ and $O(n^3\alpha(4n, n))$ for $k = 3$, which improve previously known results by $O(n/\alpha(3n, n))$ and $O(n/\alpha(4n, n))$ factors, respectively, where α is a functional inverse of Ackermann's function. The algorithms can also be implemented on a CREW PRAM in which case they need $O(\log n \log\log n)$ time using $O(m + n^2/\log n)$ processors, and $O(\log n \log\log n)$ time using $O(n^3/\log n)$ processors, respectively.
Keywords: *Combinatorial algorithms, minimum spanning trees, most vital edges, network optimization, parallel algorithms.*

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 convenience, denote by MST(G) the MST of G, and by w(MST(G)) the total weight of MST(G) (or w(MST) for short). The problem of finding an MST in G has been well studied in the past two decades [2, 10]. The best sequential algorithm for the MST problem needs $O(m \log \beta(m, n))$ time [10], where $m = |E|$, $n = |V|$, and $\beta(m, n) = \min\{i : \log^{(i)} n \leq m/n\}$ (where $\log^{(i)}$ is the iterated logarithm so that $\log^{(i+1)} = \log\log^{(i)}$ for $i > 0$.) 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 [4] for sparse graphs, and $O(\log^2 n)$ time using $O(n^2/\log^2 n)$ processors on a CREW PRAM [3] for dense graphs, respectively. It is worthwhile to mention that there are some fast randomized sequential and parallel algorithms for the MST problem [5, 14, 6]. However the deterministic simulation of these randomized algorithms may take more time than that of the

best known deterministic sequential and parallel algorithms. Therefore, in this paper we focus only on the deterministic algorithms.

One closely related problem is the k *most vital edge problem* with respect to an MST of G (the k most vital edge problem for short) which is to find an edge subset $S^* \subseteq E$, $|S^*| = k$, such that $w(\text{MST}(G(V, E - S^*)))$ is maximized. This problem has many practical applications including robust network design and distributed computing [9, 11, 12, 13, 16]. 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, usually we assume that G is $(k + 1)$-edge connected at least if we are concerned with the k most vital edge problem.

When $k = 1$, the problem becomes finding the *single most vital edge* which has been extensively studied in the literature [11, 12, 13, 15, 19]. Hsu *et al.* [11] 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 and Katoh [13] improved Hsu *et al.*'s results by giving an $O(t_{\text{MST}} + min\{m\alpha(m, n), m + n \log n\})$ 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.* [12] 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 $N \leq (m \log m)/(n\alpha(m, n) \log(m/n))$. Shen [19] 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. Recently Liang and Shen [15] presented better NC algorithms for this problem in different PRAM models. In detail, their algorithm on the EREW PRAM requires $O(\log n \log \log n)$ time and $O(m + n^2/(\log n \log \log n))$ processors; their algorithm on the CREW PRAM has the same time complexity as Shen's algorithm [19], but uses $O(m)$ processors rather than $O(m + n^2/(\log n \log \log n))$ processors used by Shen's algorithm.

For general k, not much work has been done yet. Lin and Chen [16] have shown that a generalized version of the k most vital edge 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 this problem. Frederickson and Solis-Oba [9] 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 [20] 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. Recently, Liang and Shen [15] further improved Shen's algorithm by presenting an $O(n^{k+1})$ time algorithm for fixed k. The key ingredient

they used is called *sparse, weighted k-edge certificates*. Moreover, they also presented a parallel version of their algorithm. Note that, if k is fixed, there always exists a polynomial algorithm for the k most vital edge problem which generates exact solutions.

Obviously, it is not difficult to see that the two (three) most vital edge problem here is a special case when $k = 2$ ($k = 3$). By Liang *et al.*'s algorithm, it is easy to see that this problem can be solved in $O(n^3)$ ($O(n^4)$) time. In this paper, we first present better algorithms for these special cases. The proposed algorithms run in times $O(n^2\alpha(3n, n))$ for $k = 2$ and $O(n^3\alpha(4n, n))$ for $k = 3$, respectively, which improve the results in [15] by $O(n/\alpha(3n, n))$ and $O(n/\alpha(4n, n))$ factors respectively. These improvements are obtained by utilizing a part of the algorithm for the sensitivity analysis of minimum spanning trees by Dixon *et al.* [7]. We then show that the proposed algorithms can be parallelized easily. The corresponding parallel algorithms require $O(\log n \log\log n)$ time using $O(m + n^2/\log n)$ processors and $O(\log n \log\log n)$ time using $O(n^3/\log n)$ processors on a CREW PRAM with $k = 2, 3$.

Without loss of generality, in this paper we assume that the weight associated with each edge of G is distinct and hence the MST or the minimum spanning forest (MSF for short) of graph $G(V, E - S)$ is unique for every $S \subseteq E$.

The rest of this paper is organized as follows. In Section 2 we introduce the notion of sparse, weighted k-edge certificates. In Section 3 we present sequential and parallel algorithms for the two most vital edge problem. In Section 4 we present sequential and parallel algorithms for the three most vital edge problem. A conclusion is given in Section 5.

2 Preliminaries

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 certificate* of G [17, 18], and U'_k has the following property.

Lemma 1. *[17, 18] The graph U'_k defined above is l-edge connected if and only if G is l-edge connected at least, for any integer l with $0 \leq l \leq k$.*

Notice that Lemma 1 always holds no matter whether G is a simple graph or not. We now define the sparse, weighted k-edge certificate of G as follows.

Definition 2. [15] Let G be a weighted, undirected graph, T_1 be an MST/MSF of G, and T_i be an MST/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 MSTs/MSFs $T_1, T_2, \ldots, T_i$. The graph U_k is called the sparse, weighted k-edge certificate of G.

Lemma 3. *[15] The graph U_{k+1} defined above is $(k+1)$-edge connected if and only if G is at least $(k+1)$-edge connected.*

The proof of Lemma 3 is easy and omitted. Actually it is a corollary of Lemma 1. The key point of Liang and Shen's algorithm is the following lemma.

Lemma 4. *[15] 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 $|S| \leq k$.*

Lemma 4 says 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 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 of G. Before we proceed, we introduce the following popular lemma.

Lemma 5. *Let $G(V, E)$ be an undirected, weighted simple graph, V_1 and V_2 be an arbitrary partition of the vertex set V, $V_1 \cup V_2 = V$ and $V_i \neq \emptyset$, $i = 1, 2$. Further let $Q = (V_1 \times V_2) \cap E$. Then the minimum weighted edge in Q must be in the MST/MSF of G.*

Proof. The proof is straightforward, omitted. □

Now let T be the MST of G, and $e = (u, v)$ be a tree edge in T, the *ith minimum weighted replacement edge* $r_i(e)$ of e is defined as follows. Delete e from T. 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 $Q = (W \times (V - W)) \cap E - \{e\}$, then $r_i(e)$ is the ith minimum weighted edge in Q. Obviously $w(r_i(e)) < w(r_j(e))$ if $1 \leq i < j \leq |Q|$.

Lemma 6. *Let $E(T) = \{e_1, e_2, \ldots, e_{n-1}\}$ be the edge set of T, and $r_j(e_i)$, $e_i \in E(T)$, be defined as above, then $r_j(e_i) \in U_{k+1}$ for all i and j, $1 \leq i \leq n-1$ and $1 \leq j \leq k$.*

Proof. For any edge $e \in E(T)$ and $e = (u, v)$, if we can show that $r_l(e) \in U_{l+1}$ for all l, $1 \leq l \leq k$, the lemma follows. We now prove that $r_l(e) \in U_{l+1}$. Assume that $r_j(e)$ is the first edge which is not in U_{j+1}, i.e., $r_1(e) \in U_2$, $r_2(e) \in U_3$, $\ldots$, $r_{j-1}(e) \in U_j$ but $r_j(e) \notin U_{j+1}$. Now we consider the graph $G' = G - U_j$. Obviously the edge $r_i(e)$ has been deleted from G' for all i, assume that $r_0(e) = e$, $0 \leq i \leq j-1$. Let T_{j+1} be the MST/MSF of G' by Definition 2. Let T_v and T_u be subtrees containing v and u respectively after the deletion of e from T_1 (i.e, T). Let V_1 and V_2 be the vertex set of T_v and T_u. It is easy to see that $V_i \neq \emptyset$, $i = 1, 2$ and $V_1 \cup V_2 = V$. Let Q be an edge set in G' in which the endpoints of the edges are in V_1 and V_2 respectively. Then $r_j(e)$ is the minimum weighted edge in Q because the edges $e, r_1(e), r_2(e), \ldots, r_{j-1}(e)$ have been deleted, and $r_j(e) \notin U_j$. By Lemma 5, $r_j(e)$ must be in T_{j+1}. Therefore, it must be in $U_{j+1} = U_j \cup T_{j+1}$. □

In the following we show how to find these $r_j(e_i)$s for all i and j, $1 \leq i \leq n-1$ and $1 \leq j \leq k$.

Lemma 7. *Let T be the MST of G, and $E(T) = \{e_1, e_2, \ldots, e_{n-1}\}$ be the edge set of T. The calculation of $r_j(e_i)$ for all i and j can be done in $O(n^2)$ time, where $1 \leq i \leq n-1$, $1 \leq j \leq k$ with fixed k.*

Proof. By Lemma 6, we can use U_{k+1} instead of G to compute all $r_j(e_i)$, $1 \le i \le n-1$, $1 \le j \le k$ with fixed k. It remains to present the detailed implementation. For an edge $e = (u, v) \in E(T)$, the computation of all $r_j(e)$, $1 \le j \le k$ can be done as follows: delete the edge e from T, as discussed above, so that T becomes two subtrees containing u and v respectively. Label the vertices in each such subtree by a unique identification. This costs $O(n)$ time. Let W and $V - W$ be the vertex sets containing u and v respectively. Then we chose the jth minimum weighted edge from $Q' = (W \times (V - W)) \cap U_{k+1} - \{e\}$ which is exactly $r_j(e)$ by Lemma 6. This can be done in time $O(n)$ because $|Q'| \le |U_{k+1}| < (k+1)n$ with fixed k. Therefore it costs $O(k(k+1)n) = O(n)$ time to compute all $r_j(e)$, $1 \le j \le k$. The computation of all $r_j(e_i)$, $1 \le i \le n-1$ and $1 \le j \le k$ can be done in time $O(n^2)$. □

3 Algorithms for Finding the Two Most Vital Edges

In [15], Liang and Shen have proved the following lemma.

Lemma 8. *[15] Let T be the MST of G, and $E(T) = \{e_1, e_2, \ldots, e_{n-1}\}$ be the edge set of T. Assume that S^* is the set of k edges whose deletion results in the maximum increase in the weight of the MST of $G(V, E - S^*)$. Then $E(T) \cap S^* \neq \emptyset$.*

Incorporating Lemma 4 with Lemma 8, Liang and Shen [15] suggested an $O(n^{k+1})$ time algorithm for the k most vital edge problem with fixed k. When $k = 2$, their algorithm obviously needs $O(n^3)$ time. In the following we present a better algorithm for this special case.

Since $|S^*| = 2$, by Lemma 8, then, either (i) $|S^* \cap E(T)| = 1$; or (ii) $|S^* \cap E(T)| = 2$. We now analyze these two cases separately. Assume that all $r_j(e_i)$ have been calculated for all i and j with $1 \le i \le n-1$ and $1 \le j \le 2$.

When $|S^* \cap E(T)| = 1$, this means that only one of tree edges of the original MST is one of the two most vital edges. Let $e^* \in E(T)$ be such a tree edge, then we have

Lemma 9. *Let S^* and T defined as above. If $|S^* \cap E(T)| = 1$, i.e, $e^* \in S^* \cap E(T)$, then another most vital edge in S^* must be $r_1(e^*)$.*

Proof. Assume that $r_1(e^*) \notin S^*$, then the MST of $G(V, E - S^*)$ is $T''(V, E(T) - \{e^*\} \cup \{r_1(e^*)\})$, and the weight of T'' is $w(\text{MST}) - w(e^*) + w(r_1(e^*))$ by the definition of minimum spanning trees. Now consider a subgraph G' of G after deleting both e^* and $r_1(e^*)$ from G, then the weight of MST of G' is $w(\text{MST}) - w(e^*) - w(r_1(e^*)) + w(r_2(e^*)) > w(\text{MST}) - w(e^*) + w(r_1(e^*))$ because $w(r_2(e^*)) > w(r_1(e^*))$ by the definition in Section 2, which contradicts the assumption that $r_1(e^*)$ does not belong to S^*. □

Now we discuss the case of $|S^* \cap E(T)| = 2$, which means both of the two most vital edges are tree edges. In order to find these two edges, the approach

adopted is as follows: first, delete arbitrary two edges, e_i and e_j, from T, $i \neq j$, then recompute the MST of the remaining graph $G(V, E-\{e_i, e_j\})$ (actually we use $U_3 - \{e_i, e_j\}$ instead of $G(V, E-\{e_i, e_j\})$ by Lemma 4). Let W_{e_i,e_j} be the weight of the MST after the deletion of edges e_i and e_j from G. As a result, we chose one combination from all possible combinations of two edges among $n-1$ tree edges such that the remaining MST has the maximum weight. This combination comprises the two most vital edges. Here the key is how to calculate the MST quickly in the remaining graph after deleting two tree edges from G. By utilizing the original, partial MST tree information, Liang and Shen [15] are able to reduce the computing the MST problem to a selection problem. Therefore, their algorithm needs $O(n^3)$ time when $k = 2$ because the number of combinations is $O(n^2)$, and selecting an element from an unsorted set of $O(n)$ elements costs $O(n)$ time at each time.

In this section we present a better algorithm for this problem by utilizing a partial result of the algorithm for the sensitivity analysis of minimum spanning trees due to Dixon *et al.* [7].

Lemma 10. *Let T' be the MST of a weighted undirected graph $G'(V, E')$ and $E(T')$ be the edge set of T', then all $r_1(e_i)$, $e_i \in E(T')$, can be found in time $O(\alpha(m', n)(m'+n))$ at most, or in time $O(\alpha(m', n)\log n)$ using $O((m'+n)/\log n)$ processors on a CREW PRAM, where $1 \leq i \leq n-1$, $|V| = n$ and $|E'| = m'$.*

Proof. In the algorithm for sensitivity analysis of minimum spanning trees due to Dixon *et al.* [7], one important part is computing $r_1(e_i)$ (which was defined as $b(u_i, v_i)$ in that paper) for all $e_i = (u_i, v_i) \in E(T')$. The time upper bound of the algorithm is $O((m'+n)\alpha(m', n))$. Therefore, the time used for computing all $r_1(e_i)$ is $O((m'+n)\alpha(m', n))$ at most, $1 \leq i \leq n-1$. Later Dixon and Tarjan [8] gave a parallel version of the sensitivity analysis algorithm. Their parallel algorithm requires $O(\alpha(m', n)\log n)$ time and $O((m'+n)/\log n)$ processors on a CREW PRAM. Thus, the computation of all $r_1(e_i)$ can be done in the same amount time and using the same number of processors on this model, $1 \leq i \leq n-1$. □

Combining the discussion above and Lemmas 6, 7, 8, 9 and 10, we have the detailed algorithm in Figure 1 for the two most vital edge problem. Let the variable W_{max} contain the weight of the final minimum spanning tree in the remaining graph, and e_1^* and e_2^* be the two most vital edges.

The correctness of the proposed algorithm is justified by the lemmas and the discussion above. Now we analyze the time complexity of this algorithm.

Theorem 11. *Given a weighted undirected graph $G(V, E)$, the two most vital edge problem with respect to minimum spanning trees can be solved in time $O(n^2\alpha(3n, n))$.*

Proof. By the algorithm above, Step 1 can be done in time $O(m\log\beta(m, n)) = O(n^2)$; Step 2 is an initial assignment which can be done in $O(1)$ time; Step 3 can be done in time $O(n^2)$ by Lemma 6. Step 4 needs $O(n)$ time. Step 5 is the

Procedure **Find_Two_Vital_Edges**(G,V,E)

1. find the MST T and the sparse, weighted 3-edge certificate U_3 of G;
2. $W_{max} := w(\text{MST})$;
3. **for** each edge $e_i \in E(T)$ **do**
   compute the edge $r_j(e_i)$, $j = 1, 2$;
   /* this can be done using the subgraph U_3 of G. */
   endfor;
   /* Step 4 corresponds to Case (i) */
4. **for** each edge $e_i \in E(T)$ **do**
   if $w(\text{MST}) - w(e_i) - w(r_1(e_i)) + w(r_2(e_i)) > W_{max}$ **then**
   $W_{max} := w(\text{MST}) - w(e_i) - w(r_1(e_i)) + (w(r_2(e_i))$;
   $e_1^* := e_i$; $e_2^* := r_1(e_i)$;
   endif;
   endfor;
   /* Steps 5 and 6 correspond to Case (ii) */
5. **for** each edge e_i: $1 \leq i \leq n-1$ **do**
5.1 form a tree T_{e_i} by copying T;
5.2 update the tree T_{e_i} by replacing e_i by $r_1(e_i)$;
5.3 $W_{\text{MST}}(e_i) := w(\text{MST}) - w(e_i) + w(r_1(e_i))$;
   /* $W_{\text{MST}}(e_i)$ is used to store the weight of T_{e_i} */
5.4 **for** each edge e_j: $1 \leq j \leq n-1$ and $j \neq i$ **do**
   find the first minimum weighted replacement edge e'_j
   for e_j from T_{e_i} by Dixon et al's algorithm [7];
   endfor;
5.5 **for** each edge e_j: $1 \leq j \leq n-1$ and $j \neq i$ **do**
   if $W_{\text{MST}}(e_i) - w(e_j) + w(e'_j) > W_{\text{MST}}(e_i)$ **then**
   $W_{\text{MST}}(e_i) := W_{\text{MST}}(e_i) - w(e_j) + w(e'_j)$;
   $edge(e_i) := e_j$;
   /* $edge(e_i)$ is possibly another vital edge */
   endif;
   endfor;
   endfor;
6. **for** each edge $e_i : 1 \leq i \leq n-1$ **do**
   if $W_{\text{MST}}(e_i) > W_{max}$ **then**
   $W_{max} := W_{\text{MST}}(e_i)$;
   $e_1^* := e_i$; $e_2^* := edge(e_i)$;
   endif;
   endfor

End.

Fig. 1. Pseudocode for the two most vital edge problem

dominant step which is analyzed as follows. Step 5.1 requires $O(n)$ time; Step 5.2 costs $O(n)$ time by replacing e_i with $r(e_i)$; Step 5.3 needs constant time; Step 5.4 requires $O(n\alpha(3n, n))$ time by Lemmas 4, 6 and 10; Step 5.5 costs $O(n)$ time. Therefore Step 5 can be finished in time $O(n^2\alpha(3n, n))$. Obviously Step 6 needs $O(n)$ time at most. The algorithm thus requires $O(n^2\alpha(3n, n))$ time. □

Theorem 12. *Given a weighted undirected graph $G(V, E)$, the two most vital edge problem with respect to minimum spanning trees can be solved in time $O(\log n \log\log n)$ using $O(m + n^2/\log n)$ processors on a CREW PRAM.*

Proof. The proposed algorithm can be parallelized easily. In the following we analyze the computational complexities of this parallelization. Step 1 requires $O(\log n \log\log n)$ time and $O(m+n)$ processors on an EREW PRAM [4]. Step 2 costs $O(1)$ time. Step 3 needs $O(\log n)$ time $O(n^2/\log n)$ processors. Its implementation details are as follows. For every edge $e \in E(T)$, make a copy of T. Let this copy be T_e rooted at r, delete e from T_e, using the algorithm in [1] to label each subtree. This can be done in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM. Label the endpoints of each edge in U_3 by its subtree identification which requires $O(1)$ time and $O(|U_3|) = O(n)$ processors on a CREW PRAM. Find the minimum weighted edge and the second minimum weighted edge from the set consisting of edges in U_3 whose endpoints are labeled by different subtree identification. It is obvious that this step can be done in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM. So, finding all $r_j(e)$ requires $O(\log n)$ time using $O(n^2/\log n)$ processors on a CREW PRAM for all $e \in E(T)$ and $j = 1, 2$. Step 4 needs $O(1)$ time and $O(n)$ processors at most. The analysis of Step 5 is as follows. Step 5.1 requires $O(1)$ time and $O(n)$ processors; Step 5.2 requires $O(\log n)$ time using $O(n)$ processors on an EREW PRAM; Step 5.3 needs $O(1)$ time and $O(1)$ processors; Step 5.4 requires $O(\alpha(3n, n)\log n)$ time and $O(n/\log n)$ processors on a CREW PRAM by Lemma 10; Step 5.5 and Step 6 needs $O(1)$ time and $O(n)$ processors. Therefore, the proposed algorithm requires $O(\alpha(3n, n)\log n + \log n \log\log n) = O(\log n \log\log n)$ time using $O(m + n^2/\log n)$ processors on a CREW PRAM because $\log\log n > \alpha(3n, n)$ when n is enough large. □

4 Algorithms for Finding the Three Most Vital Edges

The idea behind the proposed algorithm for finding the three most vital edges is basically the same as that for finding the two most vital edges. In the following we present this algorithm in detail.

Recall that T is the MST of G, and $E(T) = \{e_1, e_2, \ldots, e_{n-1}\}$ is the edge set of T. Let S^* be the set of the three most vital edges. Then, by Lemma 8 we know one of the following cases must be held. (i) $|S^* \cap E(T)| = 1$; (ii) $|S^* \cap E(T)| = 2$; (iii) $|S^* \cap E(T)| = 3$. We now analyze these three cases separately. Assume that $r_j(e_i)$ has been calculated for all i and j with $1 \le i \le n-1$ and $1 \le j \le 3$.

When $|S^* \cap E(T)| = 1$, this means that only one tree edge is one of the three most vital edges. Let $e^* \in E(T)$ be such a tree edge, then we have

Lemma 13. *Let S^* be the set of the three most vital edges and T be defined as above. If $|S^* \cap E(T)| = 1$, i.e, $e^* \in S^* \cap E(T)$, then the other two most vital edges in S^* must be $r_1(e^*)$ and $r_2(e^*)$.*

Proof. The proof is similar to that used in Lemma 9, omitted. □

Next we discuss the case of $|S^* \cap E(T)| = 2$, which means two of the three most vital edges are tree edges.

Lemma 14. *Let S^* be the set of the three most vital edges and T be defined as above. Assume that $S^* \cap E(T) = \{e_1^*, e_2^*\}$. Then the third most vital edge in S^* is the minimum weighted edge in either Q_1, Q_2, or Q_3, where Q_1, Q_2, and Q_3 are the edge sets which are defined as follows. After deleting e_1^* and e_2^* from T, the vertex set V is divided into three subsets V_1, V_2 and V_3 which correspond to the three subtrees. Then $Q_1 = (V_1 \times V_2) \cap E = (V_1 \times V_2) \cap U_4$, $Q_2 = (V_2 \times V_3) \cap E = (V_2 \times V_3) \cap U_4$, and $Q_3 = (V_1 \times V_3) \cap E = (V_1 \times V_3) \cap U_4$.*

Proof. Let $e_i^{(1)}$ be the minimum weighted edge in Q_i, $i = 1, 2, 3$. In the following we only deal with the case $w(e_3^{(1)}) > \max\{w(e_1^{(1)}), w(e_2^{(1)})\}$. The other two cases $w(e_2^{(1)}) > \max\{w(e_1^{(1)}), w(e_3^{(1)})\}$ and $w(e_1^{(1)}) > \max\{w(e_2^{(1)}), w(e_3^{(1)})\}$ can be handled similarly, omitted. If $w(e_3^{(1)}) > \max\{w(e_1^{(1)}), w(e_2^{(1)})\}$, then we show that either $e_1^{(1)}$ or $e_2^{(1)}$ is one of the most vital edges. Assume neither one is the third most vital edge, then the weight of the MST of the graph after deleting all edges in S^* from G is $w(\text{MST}) - w(e_1^*) - w(e_2^*) + w(e_1^{(1)}) + w(e_2^{(1)})$. Now we consider the MST of the graph after deleting e_1^*, e_2^* and $e_1^{(1)}$ from G whose weight is $w(MST) - w(e_1^*) - w(e_2^*) + w(r_1(e_1^{(1)})) + w(e_2^{(1)})$ which is larger than that of the MST of $G(V, E - S^*)$, where $r_1(e_1^{(1)})$ is the minimum weighted replacement edge of $e_1^{(1)}$ in graph $G(V, E - \{e_1^*, e_2^*\})$ which contradicts the assumption of that S^* is the set of the three most vital edges. Therefore the third most vital edge must be either $e_1^{(1)}$ or $e_2^{(1)}$ for this case. Similarly, in the other two cases, we can show that the third most vital edge is either $e_2^{(1)}$ or $e_3^{(1)}$. The lemma thus follows. □

Last we consider the case $S^* \cap E(T) = \{e_1^*, e_2^*, e_3^*\}$. In this case the three most vital edges are tree edges. In the rest we present how to find these three edges. Assume that e_1^* and e_2^* have already been found. The question is to find e_3^*. Let $T_{e_1^*, e_2^*}$ be the MST of $G(V, E - \{e_1^*, e_2^*\})$. Then e_3^* is a tree edge in $T_{e_1^*, e_2^*}$ and $e_3^* \in E(T)$ whose deletion results in the maximum increase of the weight of the final MST. So, we can apply the algorithm of Dixon [7] to find the 1st minimum weighted replacement edge for each edge in $T_{e_1^*, e_2^*}$. The algorithm is presented in Figures 2 and 3. Let the variable W_{max} contain the weight of the final minimum spanning tree in the remaining graph, and e_1^*, e_2^* and e_3^* be the three most vital edges.

The correctness of the proposed algorithm is justified by the discussion above. Now we analyze the time complexity of this algorithm.

Procedure **Find_Three_Vital_Edges**(G,V,E)
1. find the MST T and the sparse, weighted 4-edge certificate U_4 of G;
2. $W_{max} := w(\text{MST})$;
3. **for** each edge $e_i \in E(T)$ **do**
   compute the edge $r_j(e_i)$, $j = 1, 2, 3$;
   /* this can be done using the subgraph U_4 of G. */
   endfor;
   /* Step 4 corresponds to Case (i) */
4. **for** each edge $e_i \in E(T)$ **do**
   if $w(\text{MST}) - w(e_i) - w(r_1(e_i)) - w(r_2(e_i)) + w(r_3(e_i)) > W_{max}$ **then**
   $W_{max} := w(\text{MST}) - w(e_i) - w(r_1(e_i)) - w(r_2(e_i)) + (w(r_3(e_i))$;
   $e_1^* := e_i$; $e_2^* := r_1(e_i)$; $e_3^* := r_2(e_i)$;
   endif;
   endfor;
   /* Step 5 corresponds to Case (ii) */
   /* Step 5 is shown separately in Figure 3 */
   /* Steps 6 and 7 correspond to Case (iii) */
6. **for** each edge $e_i \in E(T)$ **do**
   for each edge $e_j \in E(T)$ **do**
6.1 compute the weight $W_{T_{e_i,e_j}}$ of T_{e_i,e_j} if $i \neq j$;
6.2 find $r_1(e'_k)$ for all $e'_k \in E(T_{e_i,e_j})$, $1 \leq k \leq n-1$;
   endfor;
   endfor;
7. **for** each edge $e_i \in E(T)$ **do**
   for each edge $e_j \in E(T)$ **do**
   for each edge $e'_k \in E(T_{e_i,e_j}) \cap E(T)$ **do**
   if $W_{T_{e_i,e_j}} - w(e'_k) + w(r_1(e'_k)) > W_{max}$ **then**
   $W_{max} := W_{T_{e_i,e_j}} - w(e'_k) + w(r_1(e'_k))$;
   $e_1^* := e_i$; $e_2^* := e_j$; $e_3^* := e'_k$;
   endif;
   endfor;
   endfor;
   endfor
End.

Fig. 2. Steps 1 to 4, 6 and 7 for the three most vital edge problem

Theorem 15. *Given a weighted undirected graph $G(V, E)$, the three most vital edge problem with respect to minimum spanning trees can be solved in time $O(n^3\alpha(4n, n))$.*

Proof. By the algorithm above, Step 1 can be done in time $O(m \log \beta(m, n)) = O(n^2)$; Step 2 is an initial assignment which can be done in $O(1)$ time. Step 3 can be done in time $O(n^2)$ by Lemma 6. Step 4 needs $O(n)$ time. The analysis of Step 5 is as follows. Step 5.1 needs $O(n)$ time because $|U_4| < 4n$; Step 5.2 costs $O(n)$ time; Step 5.3 can be done in $O(n)$ time; Step 5.4 and Step 5.5 take

/* Step 5 corresponds to Case (ii) */

5. **for** each edge $e_i \in E(T)$ **do**
  for each edge $e_j \in E(T)$ **do**
    if $e_i \neq e_j$ **then**

5.1 delete e_i and e_j from T, find the minimum weighted edge $e_k^{(1)}$ from Q_k, $k = 1, 2, 3$;

5.2 re-build the MST T_{e_i,e_j} for $G(V, E - \{e_i, e_j\})$;

5.3 compute $r_1(e_k^{(1)})$ for $e_k^{(1)}$ from graph $U_4 - \{e_i, e_j\}$, $k = 1, 2, 3$;

5.4 **if** $w(e_3^{(1)}) > \max\{w(e_1^{(1)}), w(e_2^{(1)})\}$ **then**
  /* this means $e_3^{(1)}$ is not chosen as a tree edge */
  if $w(e_1^{(1)}) + w(r_1(e_2^{(1)})) > w(r_1(e_1^{(1)})) + w(e_2^{(1)})$ **then**
    $e_{temp} := e_2^{(1)}$; $W_{temp} := w(e_1^{(1)}) + w(r_1(e_2^{(1)}))$
  else $e_{temp} := e_1^{(1)}$; $W_{temp} := w(r_1(e_1^{(1)})) + w(e_2^{(1)})$;
  endif;
else
  if $w(e_2^{(1)}) > \max\{w(e_1^{(1)}), w(e_3^{(1)})\}$ **then**
    /* this means $e_2^{(1)}$ is not chosen as a tree edge */
    if $w(e_1^{(1)}) + w(r_1(e_3^{(1)})) > w(r_1(e_1^{(1)})) + w(e_3^{(1)})$ **then**
      $e_{temp} := e_3^{(1)}$; $W_{temp} := w(e_1^{(1)}) + w(r_1(e_3^{(1)}))$
    else $e_{temp} := e_1^{(1)}$; $W_{temp} := w(r_1(e_1^{(1)})) + w(e_3^{(1)})$;
    endif;
  else
    /* this means $e_1^{(1)}$ is not chosen as a tree edge */
    if $w(e_2^{(1)}) + w(r_1(e_3^{(1)})) > w(r_1(e_2^{(1)})) + w(e_3^{(1)})$ **then**
      $e_{temp} := e_3^{(1)}$; $W_{temp} := w(e_2^{(1)}) + w(r_1(e_3^{(1)}))$
    else $e_{temp} := e_2^{(1)}$; $W_{temp} := w(r_1(e_2^{(1)})) + w(e_3^{(1)})$;
    endif;
  endif;
endif;

5.5 **if** $w(\text{MST}) - w(e_i) - w(e_j) - w(e_{temp}) + W_{temp} > W_{max}$ **then**
  $W_{max} := w(\text{MST}) - w(e_i) - w(e_j) - w(e_{temp}) + W_{temp}$;
  $e_1^* = e_i$; $e_2^* := e_j$; $e_3^* := e_{temp}$;
endif;
  endif;
  endfor;
endfor;

Fig. 3. Step 5 for the three most vital edge problem

constant time. So, Step 5 takes $O(n^3)$ time. Step 6 is the dominant step which is analyzed as follows. Step 6.1 requires $O(n)$ time; Step 6.2 costs $O(n\alpha(4n,n))$ time by Lemma 10. Step 6 thus takes $O(n^3\alpha(4n,n))$ time. Step 7 costs $O(n^3)$ time. The algorithm thus requires $O(n^3\alpha(4n,n))$ time. □

Theorem 16. *Given a weighted undirected graph $G(V,E)$, the three most vital edge problem with respect to minimum spanning trees can be solved in time $O(\log n \log\log n)$ using $O(n^3/\log n)$ processors on a CREW PRAM.*

Proof. The proposed algorithm can be parallelized easily. In the following we analyze the computational complexities of this parallelization. Step 1 requires $O(\log n \log\log n)$ time and $O(m+n)$ processors on an EREW PRAM [4]. Step 2 costs $O(1)$ time. Step 3 needs $O(\log n)$ time $O(n^2/\log n)$ processors. Its implementation details are as follows. For every edge $e \in E(T)$, make a copy of T. Let this copy be T_e rooted at r, delete e from T_e, using the algorithm in [1] to label each subtree. This can be done in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM. Label the endpoints of each edge in U_4 by its subtree identification which requires $O(1)$ time and $O(|U_4|) = O(n)$ processors on a CREW PRAM. Find the 1st, 2nd and 3rd minimum weighted edges from the set consisting of edges in U_4 whose endpoints are labeled by different subtree identification. It is obvious that this step can be done in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM. So, finding all $r_j(e)$ requires $O(\log n)$ time using $O(n^2/\log n)$ processors on a CREW PRAM for all $e \in E(T)$ and $j = 1,2,3$. Step 4 needs $O(1)$ time and $O(n)$ processors at most. The analysis of Step 5 is as follows. Step 5.1 requires $O(\log n)$ time and $O(n/\log n)$ processors. The implementation details are similar to that used in Step 3; Step 5.2 requires $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM; Step 5.3 requires $O(\log n)$ time using $O(n/\log n)$ processors on a CREW PRAM; Step 5.4 and Step 5.5 need $O(1)$ time and $O(1)$ processors. Therefore Step 5 needs $O(\log n)$ and $O(n^3/\log n)$ processors on a CREW PRAM. Step 6 is a dominant step. Step 6.1 requires $O(\log n)$ time and $O(n/\log n)$ processors on an EREW PRAM; Step 6.2 runs in $O(\alpha(4n,n)\log n)$ time using $O(n/\log n)$ processors on a CREW PRAM by Dixon *et al.*'s algorithm [8]. Step 6 thus requires $O(\alpha(4n,n)\log n)$ time using $O(n^3/\log n)$ processors on a CREW PRAM. Step 7 requires $O(1)$ time and $O(n^3)$ processors on a CREW PRAM, or $O(\log n)$ time using $O(n^3/\log n)$ processors on the same model by Brent's Theorem. Therefore, the proposed algorithm requires $O(\alpha(4n,n)\log n + \log n \log\log n)$ $= O(\log n \log\log n)$ time using $O(n^3/\log n)$ processors on a CREW PRAM because $\log\log n > \alpha(4n,n)$ when n is enough large. □

5 Conclusions

In this paper better algorithms for finding the k most vital edges with respect to minimum spanning trees have been suggested for $k = 2$ and $k = 3$. The proposed algorithms run in times $O(n^2\alpha(3n,n))$ and $O(n^3\alpha(4n,n))$ respectively, which improve previously known results by $O(n/\alpha(3n,n))$ and $O(n/\alpha(4n,n))$ factors,

respectively. We also show that the proposed algorithms can be parallelized easily. Actually a straightforward generalization of the algorithms in this paper can be used to solve the k most vital edge problem in time $O(n^k\alpha((k+1)n, n))$ for any fixed k.

Acknowledgments

We thank Brendan McKay for his valuable comments. We also thank the anonymous referees for carefully reading the paper and making useful suggestions. The authors were partially supported by the Australian Research Council.

References

1. K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka. A simple parallel tree contraction algorithm. *J. Algorithms*, 10, 1989, 287–302.
2. A. Aho, J. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison–Wesley, 1974.
3. F. Y. Chin, J. Lam, and I -N Chen. Efficient parallel algorithms for some graph problems. *Commun. of ACM.*, 25, 1982, 659–665.
4. K. W. Chong and T. W. Lam. Finding minimum spanning trees on the EREW PRAM. *Manuscript*, Feb., 1996.
5. R. Cole, P. N. Klein, and R. E. Tarjan. A linear work parallel algorithm for minimum spanning trees. In *Proc. of the 6th Annual ACM Symp. on Parallel Algorithms and Architectures*, June, 1994, 11–15.
6. R. Cole, P. N. Klein, and R. E. Tarjan. Finding minimum spanning forests in logarithmic time and linear work. In *Proc. of the 8th Annual ACM Symp. on Parallel Algorithms and Architectures*, June, 1996, 243–250.
7. B. Dixon, M. Rauch and R. E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM J. Comput.*, 21, 1992, 1184–1192.
8. B. Dixon and R. E. Tarjan. Optimal parallel verification of minimum spanning trees in logarithmic time. *Lecture Notes in Computer Science*, 805, 1994, 13–22.
9. G. N. Frederickson and R. Solis-Oba. Increasing the weight of minimum spanning trees. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, January, 1996, 539–546.
10. H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2), 1986, 109–122.
11. L. Hsu, R. Jan, Y. Lee, and C. Hung. Finding the most vital edge with respect to minimum spanning tree in a weighted graph. *Inform. Proc. Lett.*, 39, 1991, 277–281.
12. L. Hsu, P. Wang, and C. Wu. Parallel algorithms for finding the most vital edge with respect to minimum spanning tree. *Parallel Computing*, 18, 1992, 1143–1155.
13. K. Iwano and N. Katoh. Efficient algorithms for finding the most vital edge of a minimum spanning tree. *Inform. Proc. Lett.*, 48, 1993, 211-213.
14. D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. In *J. of the ACM*, 42, 1995, 321–328.
15. W. Liang and X. Shen. Finding the k most vital edges in the minimum spanning tree problem. To appear in *Parallel Computing*, 1997.

16. K. Lin and M. Chern. The most vital edges in the minimum spanning tree problem. *Inform. Proc. Lett.*, 45, 1993, 25–31.
17. H. Nagamochi and T. Ibaraki. A linear time algorithm for finding a sparse k-connected subgraph of a k-connected graph. *Algorithmica*, 7, 1992, 583–596.
18. H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discrete Math.*, 5, 1992, 54–66.
19. 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.
20. 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.

This article was processed using the LaTeX macro package with LLNCS style