Improved Parallel Algorithms for Finding the Most Vital Edge of A Graph with Respect to Minimum Spanning Tree*

Hong Shen
School of Computing and Information Technology
Griffith University
Nathan, QLD 4111, Australia
hong@cit.gu.edu.au

Abstract

Let G be a connected, undirected and weighted graph with n vertices and m edges. A most vital edge of G with respect to minimum spanning tree is an edge whose removal from G will cause the greatest weight-increase in the minimum spanning tree of the remaining graph. This paper presents fast parallel algorithms that compute the most vital edge of G in $O(\log n)$ time using $O(m \log\log\log n/\log n + n)$ CRCW processors, and in $O(\log n \log\log n)$ time using $O((m + n^2/\log n)/\log\log n)$ CREW processors, respectively. This improves the known results of $O(\log n)$ time and $O(m)$ processors on CRCW PRAM [11, 13], and of $O(n)$ time and $O(n^2/\log^2 n)$ processors on CREW PRAM [13].

Keywords: Minimum spanning tree, most vital edge, parallel algorithm, PRAM.

1 Introduction

Let $G = (V, E)$ be a connected, undirected and weighted graph with vertex set V and edge set E, where $|V| = n$ and $|E| = m$. Associated with each edge e there is a real valued weight $w(e)$. Let MST_G be a minimum spanning tree of G and $w(MST_G)$ be the total (edge) weight of MST_G. Denote the resulting graph from deleting edge e from G by $G - e$. The *most vital edge* of G with respect to (w.r.t.) minimum spanning tree is defined as an edge e such that $w(MST_{G-e}) \leq w(MST_{G-e'})$ for every $e' \in E$.

The problem of finding the most vital edge has been studied extensively due to its important applications in network design and reliability issues [4, 5, 6]. Several sequential and parallel solutions to the most vital edge problem have been proposed recently [4, 5, 7, 11, 13]. On parallel algorithms, Hsu et al. first proposed two algorithms running in $O(n^{1+\epsilon})$ time using $n^{1-\epsilon}$ processors and $O(m \log \frac{m}{N}/N + n\alpha(m, n) \log \frac{m}{n})$ time using $N \leq m \log m/(n\alpha(m, n) \log \frac{m}{n})$ processors on CRCW PRAM, respectively [5]. Suraweera and Maheshwari given a different (though slower) CRCW algorithm running in $O(m)$ time using $m + n$ processors [12]. The first fast parallel algorithm was proposed by Katajainen and träff which runs in $O(\log n)$ time using m CRCW processors, and hence in $O(\log^2 n)$ time with m CREW processors [11]. The same

*This work was done while the author was visiting Department of Computer Science, Åbo Akademi University, Finland.

result was also achieved by Suraweera et al. [13]. In addition, a CREW algorithm running in $O(n)$ time with $O(n^2/\log^2 n)$ processors was also presented in [13].

In this paper, we improve the known results by giving

(a) CRCW algorithm running in $O(\log n)$ time using $O(m \log\log\log n/\log n + n)$ processors, and

(b) CREW algorithm running in $O(\log n \log\log n)$ time using $O((m + n^2/\log n)/\log\log n)$ processors.

We use the same MINIMUM-CRCW PRAM as used in [11, 13] which allows only the processor holding the minimum value (data) to write when multiple processors try to write to the same memory location.

2 Edge replacement in minimum spanning tree

Assume that G is at least 2-edge connected so that it contains no bridges. Otherwise, we simply find a bridge and report it as a most vital edge, which can be done in $O(m)$ time. For simplicity and without loss of generality, we make the same assumption as in [5, 6, 13] that all the edge-weights are distinct. Clearly under this assumption, a minimum spanning tree and hence most vital edge of G is unique.

The following lemma was given by Hsu et al. in [4]:

Lemma 1 *If e is a most vital edge of G then e is an edge of MST_G.*

By the definition of most vital edge and Lemma 1, a most vital edge can be computed by finding an edge in a minimum spanning tree of G whose replacement with a non-tree edge will cause the greatest weight increase. Define the replacement edge of $e \in MST_G$, denoted by $r(e)$, to be an edge in $G - MST_G$ s.t. $MST_G - e + r(e)$ is a minimum spanning tree of $G - e$. Finding the most vital edge of G is equivalent to finding $e \in MST_G$ s.t. $w(r(e)) - w(e) \geq w(r(e')) - w(e')$ for any $e' \neq e$ in MST_G.

The problem is now concluded to the *edge replacement* problem in minimum spanning tree, that is, how to find the replacement edge $r(e)$ for every e in MST_G. Let $MST_G = (V, T)$, $G' = G - MST_G = (V, E - T)$ and $R(e) = \{e' \mid e' \text{is a candinate for} r(e)\}$. For $e = u, v$, Tarjan first showed that $r(e)$ is an edge with minimum weight in G' that makes u and v connected after deleting e [14]. Hence $|R(e)| = |E - T| = O(m)$. Based on this fact parallel algorithms for computing the most vital edge using m processors are derived [11]. Iwano and Katoh reduced $|R(e)|$ to $O(n)$ by showing that $r(e)$ is an edge in $MaxST_G$, where $MaxST_G$ is a *maximum spanning tree* of G under weight function *new* [7]: $new(e) = 0$ for $e \in T$ and $new(e) = B - w(e)$ for $e \in E - T$ (B is a big constant greater than the maximum edge-weight).

Iwano and Katoh's lemma directly results in the following lemma which forms the base of our algorithms:

Lemma 2 *For any e in MST_G, $r(e)$ is an edge in $MST_{G'}$, where $G' = G - MST_G$.*

Proof Let T_1 and T_2 be two subtrees of MST_G after deletion of e from MST_G, $cut(T_1, T_2)$ be the set of edges across the cut between T_1 and T_2 in $G - e$, and e^* the edge with minimum weight in $cut(T_1, T_2)$. By definition it is clear that $r(e) = e^*$. We now show that $e^* \in MST_{G'}$.

Assume that $e^* \notin MST_{G'}$. Then $MST_{G'} + e^*$ must contain a cycle across between T_1 and T_2. Let $\bar{e}^* \in cut(T_1, T_2)$ is on the cycle. $MST_{G'} + e^* - \bar{e}^*$ forms another spanning tree of G', and its weight is smaller than the weight of $MST_{G'}$ since $w(e^*) < w(\bar{e}^*)$. The latter contradicts the fact that $MST_{G'}$ is a minimum spanning tree of G'. □

Given G, from the definition of replacement edge and Lemma 2, the following algorithm for computing $r(e)$ for every $e \in MST_G$ can be derived immediately:

Algorithm `MST-EdgeReplacement`(G, r)
{*Input G, output replacement-edge array r initialized with ∞.*}

Construct MST_G and $MST_{G'}$, where $G' = G - MST_G$;

for every edge $e' = (u, v) \in MST_{G'}$ **do**
  for every edge e on the path from u to v in MST_G **do**
    if $w(r(e)) > w(e')$ **then** $r(e) := e'$

end.

3 The algorithm

Based on Lemma 1 and 2, we have the following parallel algorithm for computing the most vital edge:

Algorithm `Parallel-`$MVE(G,\, MVE)$ {*Input G; output MVE, the most vital edge of G.*}

0. Construct MST_G and $MST_{G'}$, where $G' = G - MST_G$;

1. **for** every edge $e' = (u, v) \in MST_{G'}$ **do in parallel**
   assign label $(w(e'), e')$ to all edges on the path $P(u, v)$ in MST_G;

2. **for** every edge $e \in MST_G$ **do in parallel**
   compute $r(e) = e'$ s.t. $(w(e'), e')$ is a label of e and $w(e') \leq w(e'')$ for any other label $(w(e''), e'')$ assigned to e;

3. Compute $MVE = \tilde{e}$ s.t. $w(r(\tilde{e})) - w(\tilde{e}) = \max_{e \in MST_G}\{w(r(e)) - w(e)\}$ in parallel

end.

Implementation of the above algorithm on a specific parallel computation model delivers a more concrete parallel algorithm. We shall in this paper consider two types of parallel computation models, the MINIMUM-CRCW PRAM (concurrent read and concurrent write with minimum-value write write-conflict resolution policy) and the CREW PRAM (concurrent read and exclusive write).

Using existing algorithms for construction of a minimum spanning tree, we can leave Step 0 alone. Implementation of the rest steps of the algorithm is realized by applying existing parallel algorithm design techniques such as list ranking and Eular tour [8].

3.1 CRCW PRAM implementation

Let MST_G be represented by an adjacency list. Given a MINIMUM-CRCW PRAM with n processors, we can implement Steps 1 and 2 in $O(\log n)$ time using the path labelling algorithm given in [11] which is based on the techniques of Eular tour (for finding the lowest common ancestor of vertex pair (u, v) which divides path $P(u, v)$ into two upwards subpaths) and pointer jumping (for propagating the label along the subpaths). Note that since we are using a MINIMUM-CRCW, when multiple processors try to write their labels to an edge $e \in MST_G$, only the one holding the label with minimum edge-weight is allowed to write, and it physically does the writing only when the edge-weight of its label is smaller than the one already assigned to e.

Step 3 can be implemented in $O(\log\log n)$ time using $O(n/\log\log n)$ CRCW processors [8].

Let $t_{MST}(m, n)$ and $p_{MST}(m, n)$ denote the time and processor complexity of constructing a minimum spanning tree of an n-vertex m-edge graph respectively. The following lemma is immediate from the above analysis:

Lemma 3 *For an n-vertex m-edge G, the most vital edge of G w.r.t. minimum spanning tree can be computed in $O(t_{MST}(m, n) + \log n)$ time using $O(p_{MST}(m, n) + n)$ MINIMUM-CRCW processors.*

We use Cole and Vishkin's algorithm [2] for constructing minimum spanning tree on CRCW in Step 0. This gives $t_{MST}(m, n) = O(\log n)$ and $p_{MST}(m, n) = O(m \log\log\log n/\log n)$. So we have the following theorem:

Theorem 1 *Given G with n vertices and m edges, we can compute the most vital edge of G w.r.t. minimum spanning tree in $O(\log n)$ time on a MINIMUM-CRCW PRAM with $O(m \log\log\log n/\log n + n)$ processors.*

3.2 CREW implementation

We now show how to implement our algorithm on a CREW PRAM with $n^2/\log n$ processors.

For Step 1, we allocate one processor to each edge in $MST_{G'}$ and use an array L of size $n-1$ at each edge in MST_G, where $L[i]$ is used for keeping the label (if there is) to be written by processor i, $1 \leq i \leq n-1$. We use the same path labelling procedure as in the CRCW algorithm, but at each step of writing different processors write their labels into different cells in array L at edge e. So this step takes $O(\log n)$ time with n processors.

For Step 2, we allocate $n/\log n$ processors to each edge in MST_G to find the label with minimum edge-weight among non-empty cells in L, thus to generate the replacement edge. This can be done easily in $O(\log n)$ time by applying standard techniques: first dividing L into $n/\log n$ subarrays of size $\log n$ and finding the local minimum of every subarray using one processor simultaneously, and then finding the global minimum among the $n/\log n$ local minima in parallel. Hence this step can be completed in $O(\log n)$ time with a total of $n^2/\log n$ processors.

For Step 3, apparently finding the maximum among $n-1$ data $(w(r(e)) - w(e)$ for all $e \in MST_G)$ can be done in $O(\log n)$ time using n processors.

Let $t'_{MST}(m, n)$ and $p'_{MST}(m, n)$ denote the time and processor complexity for constructing a minimum spanning tree of an n-vertex m-edge graph on a CREW PRAM respectively. We have the following lemma from the above analysis:

Lemma 4 *For an n-vertex m-edge G, the most vital edge of G w.r.t. minimum spanning tree can be computed in $O(t'_{MST}(m,n) + \log n)$ time using $O(p'_{MST}(m,n) + n^2/\log n)$ MINIMUM-CREW processors.*

As Karger indicated [10], Chong and Lam's EREW algorithm for computing connected components [1] can be extended for construction of minimum spanning tree (similar to Johnson and Metaxas's extension [9]). Their result gives $t'_{MST}(m,n) = O(\log n \log\log n)$ and $p'_{MST}(m,n) = O(m/\log\log n)$. This is perhaps the best deterministic algorithm in terms of worst-case performance. Matching this time complexity allows us to reduce the number of processors employed in Step 3 by a factor of $\log\log n$: Step 3 can be implemented in $O(\log n \log\log n)$ time with $O(n^2/(\log n \log\log n)$ processors by dividing L into $n/(\log n \log\log n)$ equal-sized parts instead of $n/\log n$ parts as described above. This yields the following theorem:

Theorem 2 *Given G with n vertices and m edges, we can compute the most vital edge of G w.r.t. minimum spanning tree in $O(\log n \log\log n)$ time on a CREW PRAM with $O((m + n^2/\log n)/\log\log n)$ processors.*

4 Concluding remarks

We have presented an efficient parallel algorithm for computing the most vital edge w.r.t. minimum spanning tree in a graph with n vertices and m edges that runs in $O(\log n)$ time on a MINIMUM-CRCW PRAM with $O(m \log\log\log n/\log n + n)$ processors and in $O(\log n \log\log n)$ time on a CREW PRAM with $O((m + n^2/\log n)/\log\log n)$ processors respectively. Our algorithm improves the previously known results on the most vital edge problem which require $O(\log n)$ time and $O(m)$ CRCW processors [11, 13] (and hence $O(\log^2 n)$ time and $O(m)$ CREW processors by simulating CRCW on CREW), $O(n)$ time and $O(\frac{n^2}{\log^2 n})$ CREW processors [13] respectively.

Clearly the cost of our algorithm is dominated by that required for constructing a minimum spanning tree.

Our algorithm is based on an efficient method of finding the replacement edge for each edge in a minimum spanning tree in $O(n)$ sequential time. It applies existing parallel algorithm design techniques for fast parallelization of the method.

As shown in [3, 11, 14, 15], finding minimum replacement edges for all edges in a minimum spanning tree can lead directly to the solutions to verification of a minimum spanning tree and sensitivity analysis of minimum spanning trees. It is not hard to see that our algorithm can be easily modified to the algorithms for minimum spanning tree verification and for computing the sensitivity for all tree-edges within the same cost.

Acknowledgement

The author wishes to thank Ralph-Johan Back for his support during the course of this work.

References

[1] K. W. Chong and T. W. Lam. Improved parallel algorithm for finding connected components. In *Proc. IEEE 1st Intern Conf. on Algorithms and Architectures for Parallel Processing, Brisbane*, pages 452–459, 1995.

[2] R. Cole and U. Vishkin. Approximate and exact parallel scheduling with applications to list, tree and graph problems. In *27th IEEE Symposium on Foundations of Cmputer Science*, pages 478–491, 1986.

[3] B. Dixon and R. E. Tarjan. Optimal parallel verification of minimum spanning trees in logarithmic time. In *Canada-France Conference on Parallel and Distributed Computing, Theory and Practice*, 1994.

[4] L. H. Hsu, R. H. Jan, Y. C. Lee, C. N. Hung, and M. S. Chern. Finding the most vital edge with respect to minimum spanning tree in weighted graphs. *Information Processing Letters*, 39:277–281, 1991.

[5] L. H. Hsu, P. F. Wang, and C. T. Wu. Parallel algorithms for finding the most vital edge with respect to minimum spanning tree. *Parallel Computing*, 18:1143–1155, 1992.

[6] K. Iwano and N. Katoh. Efficient algorithms for finding the most vital edge of a minimum spanning tree. *Information Processing Letters*, 48, 1993.

[7] K. Iwano and N. Katoh. Efficient algorithms for finding the most vital edge of a minimum spanning tree. *Information Processing Letters*, 48:211–213, 1993.

[8] J. Jájá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[9] D. B. Johnson and P. Metaxas. A parallel algorithm for the minimum spanning tree. In *Proc. 4th Annual ACM-SIAM Symp. Parrallel Algorithms and Architectures*, 1992.

[10] D. R. Karger. Random sampling in matroids, with applications to graph connectivity and minimum spanning trees. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1993.

[11] J. Katajainen and J. L. Träff. Simple parallel algorithms for the replacement edge problem and related problems on minimum spanning trees. Technical Report DIKU-94/18, Department of Computer Science, University of Copenhagen, 1994.

[12] F. Suraweera and P. Maheshwari. A parallel algorithm for the most vital edge problem on the crcw-simd computational model. In *Proc. 17th Annual Computer Science Conf., Christchurch*, 1994.

[13] F. Suraweera, P. Maheshwari, and P. Bhattacharya. Optimal algorithms to find the most vital edge of a minimum spanning tree. Technical Report CIT-95-21, School of Computing and Information Technology, Griffith University, 1995.

[14] R. E. Tarjan. Applications of path compression on balanced trees. *Journal of ACM*, 26:690–715, 1979.

[15] R. E. Tarjan. Sensitivity analysis of minimum spanning trees and shortest path trees. *Information Processing Letters*, 14, 1982.