NC Algorithms for the Single Most Vital Edge Problem with Respect to All Pairs Shortest Paths

Sven Venema, Hong Shen, and Francis Suraweera

School of Computing and Information Technology, Griffith University
Nathan, QLD 4111, Australia

Abstract. For a weighted, undirected graph $G = (V, E)$ where $|V| = n$ and $|E| = m$, we examine the single most vital edge with respect to two measurements related to all-pairs shortest paths (APSP). The first measurement considers only the impact of the removal of a single edge from the APSP on the shortest distance between each vertex pair. The second considers the total weight of all the edges which make up the APSP, that is, calculate the sum of the distance between each vertex pair after the deletion of a tree edge. We give a sequential algorithm for this problem, and show how to obtain an NC algorithm running in $O(\log n)$ time using mn^2 processors and $O(mn^2)$ space on the MINIMUM CRCW PRAM. Given the shortest distance between each pair of vertices u and v, the *diameter* of the graph is defined as the longest of these distances. The Most vital edge with respect to the diameter is the edge lying on such a $u - v$ shortest path which when removed causes the greatest increase in the diameter. We show how to modify the above algorithm to solve this problem using the same time and number of processors. Both algorithms compare favourably with the straightforward solution which simply recalculates the all pairs shortest path information.

1 Introduction

Let $G = (V, E)$ be an undirected, weighted graph which has a non-negative weight function $d(i, j)$ associated with each edge $(i, j) \in E$. The *All Pairs Shortest Path* (APSP) problem is to find the shortest distance between each pair of vertices u,$v \in V$ in G, denoted by $dist(u, v)$. We assume that the APSP information is stored as n shortest path trees, each rooted at a distinguished vertex $\omega \in V$. The APSP problem has been extensively studied, see for example [3] for details.

The *Most Vital Edge* with respect to some specified property of a graph, in this case the APSP, is the edge which when removed causes the greatest damage to that property. We assume without loss of generality that s and t are at least 2-edge connected. If s and t are not 2-edge connected, then any edge whose removal results in the disconnection of s from t is a single most vital edge of G with respect to both MVE1 and MVE2 defined below. For the APSP problem we define two measurements of *Most Vital*.

MVE1. An edge e^* is most vital if for all other $e \in E$, $dist_{G^*}(i,j)$ - $dist_G(i,j) \geq dist_{G'}(i,j)$ - $dist_G(i,j)$, where $i,j \in V$, $G^* = (V, E \setminus \{e^*\})$ and $G' = (V, E \setminus \{e\})$.

MVE2. An edge e^* is most vital if for all $e \in E$, $S(G = (V, E \setminus \{e^*\}))_{T(\omega)} \geq S(G = (V, E \setminus \{e\}))_{T(\omega)}$, where $S(G = (V, E \setminus \{e\}))_{T(\omega)}$ is the total weight of the APSP in G with e removed, and $T(\omega)$ is the shortest path tree rooted at ω connecting all vertices.

The *diameter* of a graph $G = (V, E)$ is defined as the maximum length of the shortest paths among all pairs of vertices in V (see for example [2, 4]). For the diameter problem, we define the most vital edge as the edge which when removed causes the greatest increase in the length of the diameter.

In this paper we present an $O(\log n)$ time algorithm for both MVE1 and MVE2 using mn^2 CRCW processors. In section 2 we give the observations, lemmas and proofs used in the rest of the paper. In section 3 we develop sequential algorithms for both MVE1 and MVE2, which we use as a basis for the CRCW parallel algorithm in Section 4. We give the NC algorithm with $O(\log n)$ time complexity using mn^2 CRCW processors in Section 4. We also show how to modify this algorithm to solve the diameter problem. To the best of our knowledge, these are the first NC algorithms to solve these problems.

2 Preliminaries

Let $T(\omega)$ be the shortest path tree in G from ω to all the nodes, let $u_i(k)$ be the shortest distance from i to k, and let $v_j(k)$ be the shortest distance from k to j. Let $T(\omega)_\omega$ and $T(\omega)_{Z_e}$ be the two subtrees created by the removal of an edge e from $T(\omega)$. Note that $T(\omega)_\omega$ contains ω, and $T(\omega)_{Z_e}$ contains the set $Z_e \subseteq V$ which is the complement of $T(\omega)_\omega$.

We define a path $P_\omega(t)$ from ω to a vertex t in $T(\omega)$ to be a sequence of vertices $\omega = v_1, v_2, \ldots, v_x = t$ such that there is an edge from node v_i to v_{i+1} for $1 \leq i < x$, $x \leq n$, and no vertex is repeated. The nearest common ancestor $nca(x,y)$ of two vertices x and y in $T(\omega)$ is defined as the first common ancestor of both x and y in $T(\omega)$. If x is the parent of y, then $nca(x,y) = x$. We have the following lemma that extends the result of [10] to calculate the most vital edge with respect to the one-to-all shortest paths.

Lemma 1. An edge (x,y) is on the tree path from ω to a vertex z in a directed tree T rooted at ω if and only if the nearest common ancestor of y and z is y.

Proof. By definition.

Clearly, a most vital edge with respect to either property MVE1 or MVE2 must belong to the set of edges in $\bigcup T_\omega$ for each $\omega \in V$.

The following result was identified in [7].

Observation 1. An edge (i,j) is on a shortest ω-t path if and only if

$$u_\omega(t) = u_\omega(i) + d(i,j) + v_t(j) = \mathbf{min}_{(x,y)\in E}\{u_\omega(x) + d(x,y) + v_t(y)\}. \quad (1)$$

An erroneous claim made in [7] was corrected by Bar Noy *et al.* [1]. The corrected claim is as follows:

Observation 2. Let $T(\omega)$ be a tree of shortest paths from ω to all the nodes and let P_t be the shortest ω-t path in $T(\omega)$. If some edge $e \in P_t$ is removed from $T(\omega)$, dividing the node set V into V_ω and $\overline{V_\omega}$ such that $\omega \in V_\omega$ and $t \in \overline{V_\omega}$, then there exist shortest paths from all other nodes in $\overline{V_\omega}$ to t that do not use the edge e.

Let $Q(i,j) = \{(x,y) \in E \setminus (i,j) : x \in T(\omega)_\omega$ and $y \in T(\omega)_t\}$. It is clear that any path from ω to t in the absence of (i,j) must have an edge in this cut $Q(i,j)$. It is shown in [7] that using Observations 1 and 2, we may find this edge and the length of the shortest $\omega - t$ path in $G(V, E \setminus \{i,j)\})$ by computing the following formula.

$$\mathbf{min}_{(x,y)\in E}\{u_\omega(x) + d(x,y) + v_t(y) \mid (x,y) \in Q(i,j)\}. \quad (2)$$

From the above observations we may calculate either MVE1 or MVE2 wrt a single path from ω to t. Note that the removal of an edge e from $T(\omega)$ will create two subtrees, $T(\omega)_\omega$ and $T(\omega)_{Z_e}$. The subtree $T(\omega)_{Z_e}$ contains at most $n-1$ vertices. To reconnect these vertices to ω, we must find the replacement path from ω to every vertex in Z_e. It is not necessary to calculate the replacement path to nodes not in Z_e as the shortest path from ω to these nodes has not changed.

We make the following observation, which leads to our sequential algorithm. This algorithm lends itself to efficient parallelization.

Observation 3. Every non-tree edge (x,y) creates a cycle with a tree edge (i,j) if and only if, $(x,y) \in Q(i,j)$.

Proof. Consider an edge $(i,j) \in T(\omega)$, and an edge $(x,y) \notin T(\omega)$. We say an edge (x,y) *spans* $T(\omega)_\omega$ and $T(\omega)_t$ if its two endpoints reside in $T(\omega)_\omega$ and $T(\omega)_t$ respectively. Let (x,y) span $T(\omega)_\omega$ and $T(\omega)_t$ when (i,j) is removed from $T(\omega)$. We can find the *nearest common ancestor* of x and y ($NCA(x,y)$), that is, the node at which the paths from x to ω and from y to ω in $T(\omega)$ converge. $T(\omega)$ is rooted at ω, and the component $T(\omega)_\omega$ contains ω, thus, $NCA(x,y) \in T(\omega)_\omega$. Obviously there is only one edge (i,j) in $T(\omega)$ that spans $T(\omega)_\omega$ and $T(\omega)_t$. Hence, any edge (x,y) that spans $T(\omega)_\omega$ and $T(\omega)_t$ must create a cycle with (i,j). Furthermore, any edge (x,y) not spanning $T(\omega)_\omega$ and $T(\omega)_t$ does not create a cycle with (i,j) as both x and y must be in either $T(\omega)_\omega$ or $T(\omega)_t$, which implies that $NCA(x,y)$ is also in the same component. □

Venema *et al.* [10] showed that using an auxiliary graph called a *transmuter* the set of non-tree edges $Q(i,j)$ for each tree edge (i,j) can be simultaneously computed. We refer the reader to [9] for a detailed account of the transmuter.

3 Sequential algorithm

Tarjan [9] proposed an $O(m\alpha(m,n))$ algorithm using $O(m)$ space to solve the Shortest Path Sensitivity Analysis problem. That is, to find by how much the weight of each edge of G may be perturbed before the Shortest Path Tree of G changes. This algorithm calculates the minimum difference between a non-tree edge and the edges with which it creates a cycle. It does not associate the length of the shortest path from the source node s to any other node. Venema *et. al.* [10] solved the Single Most Vital Edge problem with respect to an $s-t$ shortest path using this approach.

To solve MVE1 and MVE2 we extend the result in [10] as follows. We perform a preprocessing step that associates with each edge $e \in T(\omega)$ the set Z_e of vertices to which a new path from ω must be calculated. We label each tree edge e with the vertex $v \in (V \setminus \{\omega\})$ if $e \in P_\omega(v)$ to determine which vertices to reconnect to ω when e is removed. We basically run the algorithm of [10] once for each vertex pair, for a total of n^2 times. Finally we calculate the maximum increase in any inter vertex distance for MVE1, and the change in the weight of the n APSP trees for MVE2. Note that we may use the algorithm for solving MVE1 to compute the most vital edge with respect to the diameter of the graph. We first compute the length of the replacement path for each vertex pair that is disconnected when a tree edge is removed. Then, instead of computing the change in vertex distances, we find the maximum distance between each pair of vertices. This gives us the maximum distance between any pair of vertices, and hence, the new diameter of the graph.

We first give a procedure Pre-MVE, which we use to preprocess the n all-pairs shortest path trees. This procedure is used for both MVE1 and MVE2.

Procedure Pre-MVE

Input: The shortest path tree $T(z)$ rooted at z for each $z \in V$.
Output: The sets Z_{e_ω} of nodes disconnected when e is removed from $T(\omega)$
For each $\omega \in V$ do
    For each $e \in T(\omega)$ do
        Calculate Z_{e_ω}
        { The set of vertices to reconnect with ω when
          e is removed from ω }
    End {for}
End {for}

We now give two procedures Post-MVE1, and Post-MVE2 which generate MVE1 and MVE2 respectively after all replacement paths have been calculated.

Procedure Post-MVE1

For each tree edge e do
$M(e) = \text{MAX } \{dist(\omega, j) - u_\omega(j), e \in T_\omega, \text{ and } j \in Z_e\}$
End {for}
MVE1 = MAX$(M(e))$

Procedure Post-MVE2

For each tree edge e do
$S(e) = \sum \{dist(\omega, j) - u_\omega(j), e \in T_\omega, \text{ and } j \in Z_e\}$
End {for}
MVE2 = MAX$(S(e))$

The extended algorithm from [10] now looks as follows:

Algorithm All-Pairs-MVE

1. Call Procedure Pre-MVE
2. Calculate $u(x)$ and $v(x)$ for every vertex x in V.
3. Compute an $n * n$ matrix $A(f)$ for each non-tree edge $f = (x, y)$, and label each element $[i, j]$ with $d(x, y) + u_i(x) + v_j(y)$.
4. Compute the nearest common ancestor $nca(x, y)$ for every non-tree edge (x, y).
5. Calculate the auxiliary edge f' for each non-tree edge f.
   { The auxiliary edge f' for a non-tree edge $f = (x, y)$ is defined as $(nca(x, y), y)$. }
6. Construct a transmuter for the auxiliary graph $G' = (V, E')$ where $E' = \{e \mid e \text{ is a tree edge } \} \cup \{f' \mid \text{f is a non-tree edge } \}$.
7. Label each sink f' of the transmuter with matrix $A(f)$. Each sink f' will now have n^2 labels associated with it, each of which represents a replacement path for a vertex pair.
8. Process the nodes of the transmuter in reverse topological order. For any node that is not a sink, label it with the minimum of the labels of it's immediate successors. That is, node v_i receives the minimum of the i^{th} label of each of v's immediate successors.
9. Call either procedure Post-MVE1 or Post-MVE2 to calculate MVE1 or MVE2 respectively.

We omit the complexity analysis of the sequential algorithm as the focus of this paper is the NC algorithm on the CRCW PRAM model. For a more detailed discussion see [10] and [9] We now show it's implementation on the CRCW PRAM model.

4 Parallel algorithm

The main idea of the parallel algorithm is to first calculate the set of nodes Z_e disconnected when each tree edge e is removed. Then each non-tree edge has an $n * n$ matrix of inter vertex replacement paths computed. Each element of this $n * n$ matrix is propagated up the transmuter structure. We keep only the minimum value computed so far for each element of the matrix. Once the propagating stage is complete, we use either Procedure Post-MVE1 or Procedure Post-MVE2 to solve problem MVE1 or MVE2, respectively.

Algorithm Parallel-MVE

Input: The shortest path tree $T(z)$ rooted at z for each $z \in V$.
Output: The single most vital edge wrt all pairs shortest paths
1. Compute Z_e for each tree edge $e \in E$ in parallel
   using procedure Pre-MVE
2. Compute the $n * n$ matrix $A(f)$ for each non-tree edge f
3. Using the path labeling technique from [6] and the construction in
   Section 3, compute the $|V|$ minimum labels for each tree edge in parallel:
   3.1 For each non-tree edge (x, y), project its n^2 labels to every tree
      edge on the cycle $T(\omega) \bigcup \{x, y\}$ for each $\omega \in V$.
   3.2 For each tree edge find the n^2 minimum labels among all labels
      assigned to it in Step 3.1.
4. Call either Procedure Post-MVE1 or Post-MVE2 to calculate MVE1
   or MVE2 respectively.

4.1 CRCW implementation

The computational model for the algorithm is the MINIMUM-CRCW PRAM. On this model, if a write conflict arises, the processor holding the minimum value is allowed to write. This processor writes it's value if and only if the value to be written is smaller than any previously computed value.

Given a MINIMUM-CRCW PRAM with m processors, we can implement Tarjan's Shortest Path Sensitivity Analysis algorithm [9] in $O(\log n)$ time by a path labelling technique described in [6]and later used in [8]. This combines the techniques of Euler tour and list ranking using pointer jumping [5] to find the nearest common ancestor of a non-tree edge (x, y) and assign the correct label to each edge on the paths from x and y to that ancestor. The Euler tour technique is used to split the path P from x to y into two subpaths by finding the lowest common ancestor of vertices x and y. We then use list ranking with pointer jumping to propagate the labels along the two subpaths.

In Step 1, the set Z_{e_ω} is calculated for each edge $e \in T(\omega)$, for each $\omega \in V$, by applying the Euler tour technique and lemma 1 as follows;

1.1. Root $T(\omega)$ at ω using the Euler tour technique (see [5, Section 3.2])
1.2. For each $v \in V$ do
    For each $w \in V \setminus \{\omega\}$ do
        If $nca(v, w) = w$ Then
            Set position v at edge $(parent(w), w)$ to a 1.
        End { if }
    End { for }
End { for }

Rooting a tree using the Euler tour technique requires $O(\log n)$ time using n EREW processors. There are n vertices, each of which requires n nearest common ancestor calculations. An edge can appear in at most n shortest path trees. None of the above steps require concurrent read or write, thus, we can implement step 1 in $O(\log n)$ time using n^3 EREW processors.

Step 2 may be done in constant time on mn^2 processors as there are at most $m - n + 1$ non-tree edges, and each non-tree edge requires n^2 constant-time path calculations. Step 3 can be done in $O(\log n)$ time using mn^2 processors (for details see [10]).

If we calculate MVE1, Procedure Post-MVE1 can be done in constant time on mn^2 processors as there are at most m distinct shortest path tree edges, and we calculate the minimum of at most n^2 elements for each of these. For procedure Post-MVE2 we compute the sum of at most n^2 elements for each tree edge. There are at most m such tree edges. Hence, this step can be done in $O(\log n)$ time using mn^2 processors.

References

1. A. Bar-Noy, S. Khuller, and B. Schieber. The complexity of finding most vital arcs and nodes. Technical Report CS-TR-3539, University of Maryland Institute for Advanced Studies, College Park (1995).
2. W.G. Brown. Reviews in graph theory. American Mathematical Society, Providence, RI (1980).
3. N. Deo and A. Pan. Shortest Path Algorithms: Taxonomy and Annotation. Networks 14 (1984), 275–323.
4. J. Ho, D.T. Lee, C. Chang, and C.K. Wong. Minimum diameter spanning trees and related problems. SIAM J. Comput. Vol. 20 No. 5 (1991), 987–997.
5. J. Jájá. *An Introduction to Parallel Algorithms.* Addison-Wesley, Reading, Mass., 1992.
6. 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.
7. K. Malik, A.K. Mittal and S.K. Gupta. The k most vital edges in the shortest path problem. *Oper. Res. Lett. 8* (1989), 223–227.

8. H. Shen. Improved parallel algorithms for finding the most vital edge of a graph with respect to a minimum spanning tree. Technical Report, School of Computing and Information technology, Griffith University, Australia, URL: http://www.cit.gu.edu.au/research/reports (1996).
9. R.E. Tarjan. Sensitivity analysis of minimum spanning trees and shortest path trees. *IPL 14* (1982), 30–33.
10. S. Venema, H. Shen and F. Suraweera, NC Algorithms for the Single Most Vital Edge Problem with Respect to Shortest Paths, *IPL 60* (1996), 243–248.

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