July 1988

UILU-ENG-88-2233
ACT-93

COORDINATED SCIENCE LABORATORY
College of Engineering
Applied Computation Theory

FINDING ALL MINIMUM SIZE SEPARATING VERTEX SETS IN A GRAPH

Arkady Kanevsky

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-88-2233 ACT #93	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Semiconductor Research Corporation
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) P.O. Box 12053 Research Triangle Park, NC 27709
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Semiconductor Research Corp. 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER 87-DP-109
8c. ADDRESS (City, State, and ZIP Code) P.O. Box 12053 Research Triangle Park, NC 27709	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Finding All Minimum Size Separating Vertex Sets in a Graph

12. PERSONAL AUTHOR(S)
Kanevsky, Arkady

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	1988 July 6	16

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	network reliability, graph theory, separating sets, algorithms

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

We present a new algorithm based upon network flows for finding all minimum size separating vertex sets in an undirected graph. The sequential implementation of our algorithm runs in $\Theta(\min(\max(Mnk, knm\min(k,\sqrt{n})), \max(Mn, k^2n^3))) = O(2^k n^3)$ time, where M is the number of minimum size separating vertex sets The parallel implementation runs either in $O(k\log n)$ deterministic time using $\Theta(M^2n^2 + knN^\alpha) = O(4^k \frac{n^6}{k^2})$ CRCW PRAM processors or in $O(\log^2 n)$ randomized time using $\Theta(M^2n^2 + kn^2N^\alpha) = O(4^k \frac{n^6}{k^2})$ processors on a CRCW PRAM, where n is the number of vertices in the graph and k is the connectivity of the graph, and N^α is the number of processors needed for parallel matrix multiplication.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

Finding All Minimum Size Separating Vertex Sets in a Graph

Arkady Kanevsky

Coordinated Science Laboratory
University of Illinois
Urbana, IL 61801

June 1988

ABSTRACT

We present a new algorithm based upon network flows for finding all minimum size separating vertex sets in an undirected graph. The sequential implementation of our algorithm runs in $\Theta(\min(\max(Mnk, knm\min(k,\sqrt{n})), \max(Mn, k^2n^3))) = O(2^k n^3)$ time, where M is the number of minimum size separating vertex sets The parallel implementation runs either in $O(k\log n)$ deterministic time using $\Theta(M^2n^2 + knN^\alpha) = O(4^k \frac{n^6}{k^2})$ CRCW PRAM processors or in $O(\log^2 n)$ randomized time using $\Theta(M^2n^2 + kn^2N^\alpha) = O(4^k \frac{n^6}{k^2})$ processors on a CRCW PRAM, where n is the number of vertices in the graph and k is the connectivity of the graph, and N^α is the number of processors needed for parallel matrix multiplication.

1. Introduction

Connectivity is one of the fundamental graph properties, and there has been a considerable amount of work on algorithms and structural aspects of this property. Applications of this problem arise in operation research for scheduling problems, network analysis in electrical engineering and many other real-life problem. The most direct application of this problem is for the reliability of networks [Ba,BaPr,Bu,Co,RaCo,Ro].

There are well-known sequential linear-time algorithms for determining vertex connectivity and biconnectivity (see e.g., [Ev1]), as well as triconnectivity [HoTa,MiRa2]. These algorithms use either depth-first search technique [Ev1,HoTa,Ta] or ear-decomposition technique [Wh,Lo,MaScVi,MiRa1]. The best deterministic sequential algorithm for testing graph 4-connectivity has time complexity $O(n^2)$ [KanRa1] and based upon ear-decomposition

---

This research was supported by the Semiconductor Research Corporation under 87-DP-109.

technique. There is a sequential $O(\max(k,n^{1/2})kmn^{1/2})$ time algorithm for determining the connectivity of a graph which is based upon network flow [EvTa,Ev2,Ga,GiSo]. We also note there are some randomized algorithms for testing k-connectivity for $k > 3$ [BeX,LiLoWi]; the running times of these algorithms are $O(n^{5/2} + nk^{5/2})$ [LiLoWi], and $O(n^{3/2}m)$ [BeX].

Efficient parallel algorithms have been designed for determining graph connectivity for small k. Clearly, there are NC algorithms for testing graph k-connectivity for any fixed k. Simply, remove all k vertex subsets of the graph and test for graph connectivity. The best parallel algorithms for graph k-connectivity for $k = 1,2$ are the efficient $O(\log n)$ parallel time algorithms using $O(m+n)$ processors on CRCW PRAM [ShVi,TaVi], for $k = 3$ an $O(\log n)$ parallel time algorithm using $O((m+n)\log n)$ CRCW PRAM processors [MiRa2,RaVi] and for $k = 4$ an $O(\log^2 n)$ parallel time algorithm using $O(n^2)$ CRCW PRAM processors [KanRa1]. There are no efficient deterministic parallel algorithm for determining the connectivity of a graph for general k.

The other question which often raised with connectivity is to find all minimum size separating vertex sets. This idea lies in the heart of the algorithms for determining graph k-connectivity for $k = 1,2,3,4$. The algorithms for graph (one)-connectivity, biconnectivity, triconnectivity and 4-connectivity find all articulation points, separating pairs, separating triplets of a graph in order to determine that a graph does not have a higher connectivity. The number of separating k-sets in a k-connected graph is $O(n^2)$ for any fixed k [Ka1,Ka2]. Furthermore, there is an $O(n)$ representation for separating pairs in a biconnected graph [KanRa2] and there is an $O(k^2n)$ compact representation for separating k-sets in a k-connected graph [Ka2].

We present an algorithm for finding all minimum size vertex separating sets for general k. The rest of this paper is organized as follows. Section 2 gives graph-theoretic definitions. In section 3 we describe the parallel model of computation we use. Section 4 presents the sequential algorithm for finding all minimum size separating vertex sets of a graph along with its time complexity analysis. Finally, section 5 presents a parallel algorithm for this problem.

2. Graph-theoretical definitions

An *undirected graph* $G=(V,E)$ consists of a *vertex set* V and an *edge set* E containing unordered pairs of distinct elements from V. A *path* P in G is a sequence of vertices $\langle v_0, \cdots, v_k \rangle$ such that $(v_{i-1},v_i) \in E, i=1, \cdots, k$. The path P *contains* the vertices $v_0, \cdots, v_k$ and the edges $(v_0,v_1), \cdots, (v_{k-1},v_k)$ and has *endpoints* v_0, v_k, and *internal*

vertices $v_1, \cdots, v_{k-1}$. The path P is a *simple path* if $v_0, \cdots, v_{k-1}$ are distinct and $v_1, \cdots, v_k$ are distinct. P is a *simple cycle* if it is a simple path and $v_0=v_k$. A single vertex is a trivial path with no edges. We denote by $|P|$, the number of vertices contained in path P.

Let $P=<v_0, \cdots, v_{k-1}>$ be a simple path. The path $P(v_i,v_j), 0\leq i,j\leq k-1$ is the simple path connecting v_i and v_j in P, i.e., the path $<v_i,v_{i+1}, \cdots, v_j>$, if $i\leq j$ or the path $<v_j,v_{j+1}, \cdots, v_i>$, if $j<i$. Analogously, $P[v_i,v_j]$ consists of the path segments obtained when the edges and internal vertices of $P(v_i,v_j)$ are deleted from P.

Let $G=(V,E)$ be an undirected graph and let $V'\subseteq V$. A graph $G'=(V',E')$ is a *subgraph* of G if $E'\subseteq E\cap\{(v_i,v_j)\mid v_i,v_j\in V'\}$. The *subgraph of G induced by V'* is the graph $G''=(V',E'')$ where $E''=E\cap\{(v_i,v_j)\mid v_i,v_j\in V'\}$.

An undirected graph $G=(V,E)$ is connected if there exists a path between every pair of vertices in V. For a graph G that is not connected, a *connected component* of G is an induced subgraph of G which is maximally connected.

A vertex $v\in V$ is an *articulation point (a.p.)* of a connected undirected graph $G=(V,E)$ if the subgraph induced by $V-\{v\}$ is not connected. G is *biconnected* if it contains no articulation point.

Let $G=(V,E)$ be a biconnected undirected graph. A pair of vertices $v_1,v_2\in V$ is a *separating pair* for G if the induced subgraph on $V-\{v_1,v_2\}$ is not connected. G is *triconnected* if it contains no separating pair.

A triplet (v_1,v_2,v_3) of unordered distinct vertices in V is a separating triplet of a triconnected graph if the subgraph induced by $V-\{v_1,v_2,v_3\}$ is not connected. G is *four-connected* if it contains no separating triplets.

In general, undirected graph is k-connected if and only if between every pair of vertices there are k vertex disjoint paths, or alternatively, removal of any $k-1$ vertices leaves a graph connected [Ev]. The equivalence of these two definitions is the well-known Menger theorem [Me].

Let $G=(V,E)$ be a k-connected undirected graph. A set V' of unordered distinct k vertices of G is a separating k-set if the induced subgraph on $V-V'$ is not connected.

3. Parallel model of computation

The parallel model of computation that we will be using is the *PRAM* model, which consists of several independent sequential processors, each with its own private memory, communicating with one another through a

global memory. In one unit of time, each processor can read one global or local memory, execute a single RAM operation, and write into one global or local memory location.

PRAMs are classified according to restrictions on global memory access. An EREW PRAM is a PRAM for which simultaneous access to any memory location by different processors is forbidden for both reading and writing. In a CREW PRAM simultaneous reads are allowed but no simultaneous writes. A CRCW PRAM allows simultaneous reads and writes. In this case we have to specify how to resolve write conflicts. We will use the ARBITRARY model in which any one processor participating in a concurrent write may succeed, and the algorithm should work correctly regardless of which one succeeds. Of the three PRAM models we have listed, the EREW model is the most restrictive, and the ARBITRARY CRCW model is the most powerful. It is not difficult to see that any algorithm for the ARBITRARY CRCW PRAM that runs in parallel time T using P processors can be simulated by an EREW PRAM (and hence by a CREW PRAM) in parallel time $T\log P$ using the same number of processors, P.

Let S be a problem which, on an input of size n, can be solved on a PRAM by a parallel algorithm in parallel time $t(n)$ with $p(n)$ processors. The quantity $w(n)=t(n)\cdot p(n)$ represents the *work* done by the parallel algorithm. Any PRAM algorithm that performs work $w(n)$ can be converted into a sequential algorithm running in time $w(n)$ by having a single processor simulate each parallel step of the PRAM in $p(n)$ time units. More generally, a PRAM algorithm that runs in parallel time $t(n)$ with $p(n)$ processors also represents a PRAM algorithm performing $O(w(n))$ work for any processor count $P<p(n)$.

Define $\mathit{poly}\log(n)=\bigcup_{k>0} O(\log^k n)$. Let S be a problem for which currently the best sequential algorithm runs in time $T(n)$. A PRAM algorithm A for S, running in parallel time $t(n)$ with $p(n)$ processors is *efficient* if

a) $t(n)=\mathit{poly}\log(n)$; and

b) the work $w(n)=p(n)\cdot t(n)$ is $T(n)\cdot \mathit{poly}\log(n)$.

An efficient parallel algorithm is one that achieves a high degree of parallelism and comes to within a *poly*log factor of optimal speed-up. A major goal in the design of parallel algorithms is to find efficient algorithms with $t(n)$ as small as possible. The simulations between the various PRAM models make the notion of an efficient algorithm invariant with respect to the particular PRAM model used. For more on the PRAM model and PRAM algorithms, see [KarRa].

4. Sequential algorithm

In this section we describe a sequential algorithm for finding all minimum size separating vertex sets in an undirected graph $G = (V,E)$. Note that the number of separating k-sets in an undirected k-connected graph is $O(2^k \frac{n^2}{k})$ [Ka2].

First, we find the connectivity k of G using network flows [EvTa,Ga,GiSo]. The time complexity of this algorithm is $O(\max(k,n^{1/2})kmn^{1/2})$. Next, we take a subset of vertices X of G of size k and find all minimum size separating vertex sets (of size k) between pairs of the form (x,v), where $x \in X$ and $v \in V$. Note the following simple observation.

Observation 1: Let $x \in X$ and $v \in V$. Assume that we have found all k-sets separating x and v in G. Then we can add edge (x,v) to E without changing (adding or deleting) any other separating k-sets of G.

Proof: This is because for any other separating k-set Y, which does not separate x and v, x and v can not belong to two different components of the graph induced by $V-Y$.

□

We repeat this process for every $x \in X$ and every $v \in V$. During this process we add at most kn edges to E. At the end of this process every vertex $x \in X$ is connected by an edge to every vertex $v \in V$. Now, if there is separating k-set Y in this graph, then $Y=X$. So, we check if X is a separating k-set of G. Every minimum size separating vertex set of G is obtained by this computation.

We note that for a given $x \in X$ we only need to conduct this procedure for these vertices $v \in V$ which are not adjacent to x. Hence, for our algorithm we choose X to be a set of k vertices of G of maximum degrees.

Algorithm

1. Find the connectivity k of G. (vertices of G are numbered from $v_1, \cdots, v_n$).
2. Find k vertices with the largest degrees $(x_1, \cdots, x_k)$.

   Check if these k vertices form a separating k-set of G

   (Let $\overline{G}$ be the directed graph which we get from G by applying the Even-Tarjan reduction (see below))

   Do $i = 1 \cdots k$

Do $j = 1 \cdots n$

3. If $v_j \neq x_i$ and v_j is not adjacent to x_i then

4. Compute a maximum flow ϕ in $\overline{G}$ from x_i to v_j

If $|\phi| = k$ then

(Find all k-sets separating x_i and v_j as follows:)

5. Construct the residual graph $\overline{G}_\phi$ of $\overline{G}$ with respect to the maximum flow ϕ

6. Shrink the strongly connected components of $\overline{G}_\phi$

7. Find all edge cutsets (closed sets) of the resulting acyclic network

For each edge cutset find the corresponding separating k-set of G

8. Add edge (x_i, v_j) to G.

enddo

enddo

The results in Picard and Queyranne [PiQu] establish the correctness of steps 5-7 for finding all separating k-sets. Let f be any maximum flow in a network N. The subset C of vertices of N is a *closed set* if and only if for every vertex $v \in C$ all of its predecessors are also members of C in N.

Lemma 1: [PiQu] A cut $(S,\overline{S})$ separating s from t in N is a minimum cut if and only if S is a closed set of N containing s but not t.

Let R be the residual graph of N with respect to the maximum flow f. Let C be a strongly connected component of R and $v \in C$. Then based upon this Lemma if $v \in S$ then C is also member of S.

Observation 2: There is one-to-one correspondence between the mincuts of $\overline{G}$ and the closed sets of N.

Definitions: Let N be the residual network of $\overline{G}$ with respect to a maximum flow. Shrink its strongly connected components into single vertices. Let L be the resulting acyclic network. (We will use L_{st} to emphasize the fact the maximum flow is taken between s'' and t').

Theorem 1: [PiQu,BaPr] The resulting acyclic network L is the same for any maximum flow.

Based upon the above Theorem 1 and Lemma 1 the problem of finding all $s-t$ mincuts in L is reduced to the problem of finding all closed sets in L which we get after shrinking all strongly connected components of N. This justifies Step 6 of the algorithm. Hence, this establishes the correctness of the algorithm by our discussion preceding the algorithm.

Let us state time complexities of all steps of the above algorithm. We establish the bounds for Steps 4 and 7 below. Step 1 takes $O(max(k,n^{1/2})kmn^{1/2})$ time [Ga,GiSo]. Step 2 takes $O(m+n)$ time. Steps 3-8 are repeated kn times. Step 4 takes $O(\min(k,\sqrt{n})(m+n))$ time and step 5 takes $O(m+n)$ time. Step 6 also takes $O(m+n)$ time. Step 7 takes $O(\min(M_{ij}n + kn^2, M_{ij}kn + n))$ time, where M_{ij} is the number of separating k-sets between v_j and x_i. Step 7 takes $O(\min(Mn + k^2n^3, Mkn + kn^2))$ time over the execution of both loops, where M is the number of separating k-sets in G. Step 8 takes constant time. The total time for the algorithm is $\Theta(\min(Mn+k^2n^3, Mkn + knm\min(k,\sqrt{n}))) = O(2^k n^3)$.

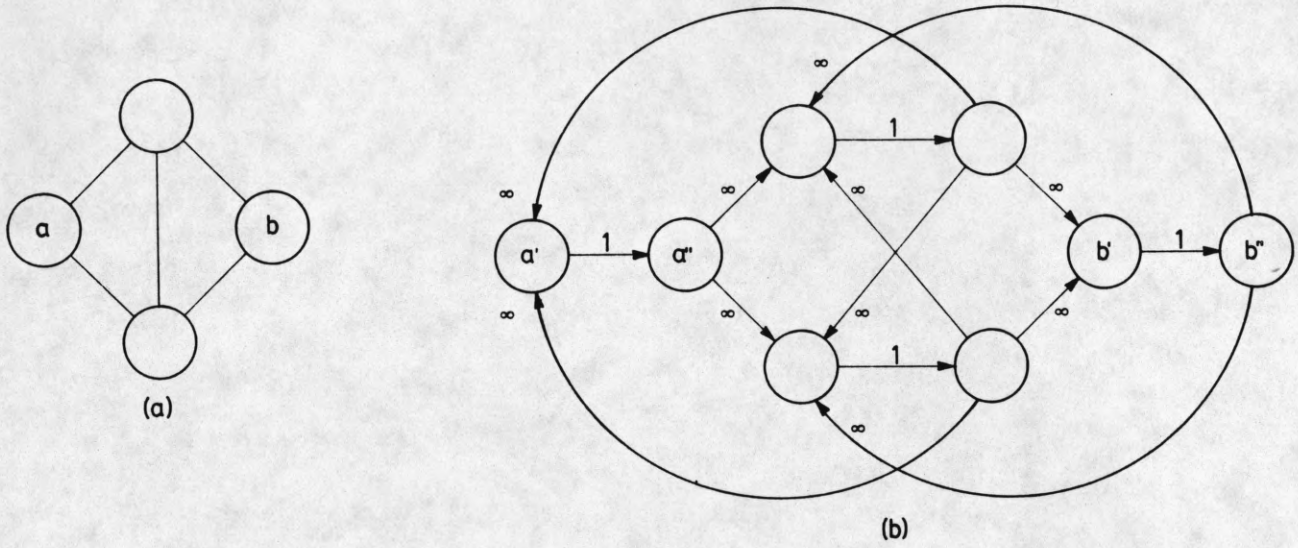
Steps 4-7 show how to find all separating k-sets between a pair of vertices s and t of G. Let us see in detail how we actually do this. First, we construct a digraph $\overline{G} = (\overline{V},\overline{E})$ as in [EvTa] (see also [Ev1]): For every vertex $v \in V$ put two vertices v' and v'' in $\overline{V}$ with a directed edge $(v',v'') \in \overline{E}$. For every edge $(u,v) \in E$, put two edges $(u'',v') \in \overline{E}$ and $(u',v'') \in \overline{E}$. Define now a network with digraph $\overline{G}$, source s'', sink t', unit capacity for all *internal* edges (edges for every vertex of G) and infinite capacity edges for all other edges of $\overline{G}$ (*external*) (see figure 1).

Lemma 2: [Ev1] If $(s,t) \notin E$ then the least cardinality (s,t) vertex separator is equal to the maximum number of vertex disjoint paths between s and t.

There are two ways to find a maximum flow from s'' to t' in $\overline{G}$. The first method is faster for small values of k, and works as follows. We find k directed paths in $\overline{G}$ from s'' to t', one path at a time. Call the resulting flow F. There are no k-sets separating s and t in G if and only if we can find a path from s'' to t' in the residual graph $\overline{G}$ with respect to F. This entire algorithm runs in $O(k(m+n))$ time.

The second algorithm is faster then the first for large values of k. In this algorithm we simply find a maximum flow in $O(m\sqrt{n})$ time using Dinic's algorithm in a unit network [Ta]. Hence we implement step 4 using one of these algorithms depending on the value of k. Hence the time complexity of this step is $O(\min(k,\sqrt{n})(m+n))$.

There are several different algorithms which find all closed sets of an acyclic directed network [BaPr,ScBa]. We will present one of them after the following lemma. Let us see know how many edges the directed acyclic



Illustrating the reduction from G to $\overline{G}$.
Figure 1.

network L can have.

Observation 3: Take any two adjacent vertices of G. There are four vertices in $\overline{G}$ corresponding to them. If the vertices of $\overline{G}$ corresponding to them were not used in a maximum flow from s'' to t' then in the residual network they will form a directed cycle of length 4 (see figure 1).

Let f be a flow from s'' to t' in $\overline{G}$, which we create by using shortest augmenting paths only. That is, we always choose a shortest augmenting path of the current residual graph to increase the flow. Let us call this flow a *shortest path flow.*

Lemma 3: Let f be a shortest path flow from s'' to t' in $\overline{G}$. Let N' be the residual graph of $\overline{G}$ with respect to f. Let L' be the acyclic graph which we get after shrinking the strongly connected components of N'. Then the number of edges in L' is $O(fn)$, where $n = |V|$. (Note that there is one path from s'' to t' in $\overline{G}$ for every unit of flow f. These paths are vertex disjoint).

Proof : We will prove that the number of edges of L' is at most $7ln$ by induction on l, where l is the number of paths in $\overline{G}$ for flow f ($f = l$).

Let $l = 1$. Take a shortest path P from s to t in G. There are no edges between vertices of the path P except the edges $E(P)$ (edges of the path itself), because the path is the shortest. A shortest path P in G corresponds to a

shortest path $\overline{P}$ in $\overline{G}$. Every edge in P corresponds to two edges in $\overline{G}$, one forward edge which is part of $\overline{P}$, and one backward edge which connects two vertices of $\overline{P}$. Also every vertex of P corresponds to an edge in $\overline{P}$. Hence, the number of edges in $\overline{G}$ corresponding to P is $3p$, where p is the length of P in G.

All vertices of $V-P$ which are not on P have at most 3 edges incident on the path. Hence, the number of edges E' between vertices of P and the vertices in $V-P$ is at most $3(n-p)$, where p is the length of P. All edges of $\overline{G}$ which correspond to $E-E'-E(P)$ in G will be shrunk in N' by Observation 3. There are 2 edges in $\overline{G}$ corresponding to each edge of E', and there is one edge in $\overline{G}$ corresponding to each vertex of G. So, the number of edges in L' is $\leq 7n$.

Assume the claim is true for $l \leq r$, and let $l = r+1$. That means that there are $O(rn)$ edges in L' for flow $f = r$. Let $P^r=(P_1, \cdots, P_r)$ be the r vertex disjoint paths in $\overline{G}$ which form the flow f. Consider the edges $\tilde{E}$ in $\overline{G}$, which neither belong to paths P^r from s'' to t' nor are adjacent to them. By the assumption there are $\leq 7rn$ edges in $E - \tilde{E}$ adjacent to paths P^r from s'' to t' or on them. Find the shortest augmentation path P in the residual graph N^* of $\overline{G}$ with respect to f from s'' to t'. An edge $e \in \tilde{E}$ will be shrunk in N' unless it belongs to P or is adjacent to it. Note that all of the neighboring edges of all previous r paths P^r were already counted by the assumption. We claim that there are at most $7n$ edges adjacent to P or on P which are in $\tilde{E}$.

Case 1. Let $e = (v_1, v_2) \in P$ such that $v_1 \notin P^r$ and $v_2 \notin P^r$. Let E_1 be the set of all edges of this type on P. Then there are at most 3 edges between each vertex of $\overline{V}$ and the endpoints of E_1 which are in $\tilde{E}$, because P is the shortest augmenting path.

Case 2. Let $e = (v_1, v_2) \in P$ such that $v_1 \in P^r$ and $v_2 \in P^r$. Then all of the edges adjacent to e were already counted by the assumption.

Case 3. Let $e = (v_1, v_2) \in P$ such that either $v_1 \in P^r$ or $v_2 \in P^r$ but not both. Note that the only edges of $\overline{G}$ which were reversed in N^* are the edges of P^r. There are two subcases

Case A. $v_1 \in P^r$. Then there is only one edge outgoing from v_2, which is the edge of Case 1. Hence, all of the edges adjacent to v_2 were already counted in Case 1, and all of the edges adjacent to v_1 were counted in Case 2.

Case B. $v_2 \in P^r$. Then there is only one edge incoming to v_1 which is the edge of Case 1. Hence, all of the edges adjacent to v_1 were already counted in Case 1, and all of the edges adjacent in v_2 were counted by Case 2.

That conclude the proof of the Claim. Hence, the number of edges in the network L is $\leq 7(r+1)n$.

This concludes the proof of the lemma.

□

Corollary: The number of edges in network L is $O(kn)$.

An *antichain* in an acyclic network is a subset of nodes R such that for all pairs of nodes i and j in R, i is neither a predecessor nor a successor of j. The algorithm Antichain finds all closed subsets of a directed acyclic network L, and runs in a linear time per subset [BaPr].

Observation 4: [PiQu,BaPr] Each antichain of L one-to-one corresponds to a closed set of L.

Let $V(L)$ be the set of vertices of L, and $E(L)$ be the set of edges of L. Let C be a part of an antichain, and the algorithm recursively adds a vertex to the antichain at each step. Initially C is empty. Let M be the number of antichains in L, initially 0.

Antichain $(V(L),E(L),C;M)$

Step 1: Choose an $i \in V(L)$ of in-degree 0 and output $C \cup \{i\}$; set $M = 1$; if $V(L)-\{i\} = \emptyset$, stop.

Step 2: Delete i from $V(L)$ to obtain $\hat{L} = (V(\hat{L}),E(\hat{L}))$ and Call Antichain $(V(\hat{L}),E(\hat{L}),C;M')$; set $M = M + M'$.

Step 3: Find all successors of i, denoted by SC(i). If $V(\hat{L})-SC(i)-\{i\} = \emptyset$, then stop; otherwise delete $SC(i) \cup \{i\}$ from $V(\hat{L})$ to obtain $\hat{L} = (V(\hat{L}),E(\hat{L}))$ and Call Antichain $(V(\hat{L}),E(\hat{L}),C \cup \{i\};M')$; set $M = M + M'$.

The correctness of this algorithm can be found in [BaPr]. The time complexity of this algorithm is $O(M_{st}m)$, where m is the number of edges in L and M_{st} is the number of k-sets separating s and t in G.

But it can be improved. Let us find all successors of each vertex of L before calling algorithm antichain. That can be done in $O(mn)$ time, where m is the number of edges in L and n is the number of vertices in L. We build a depth first search tree T_x for each vertex x of L in $O(m+n)$ time per vertex. Since L is acyclic graph we can find all successors of x in linear time from T_x. Then all substeps of Step 3 of the algorithm Antichain takes only $O(n)$ time, since we only need to read and merge the lists of maximum size n. Hence, the entire algorithm takes $O(M_{st}n + mn)$ time instead of $O(M_{st}m)$.

Recall that $m = O(kn)$ for L. The time spent by the algorithm to find all k-sets separating s and t in G is $O(\min(M_{st}kn + \min(k,\sqrt{n})kmn),(M_{st}n + k^2n^3))$. Since we add edge (x_i,v_j) to G, the separating k-sets which we have already found, cannot be found again for any other pair of vertices in the updated G. Hence,

$$\sum_{i=1}^{i=k}\sum_{j=1}^{j=n} M_{ij} = M ,$$

where M is the total number of separating k-sets in G. Since $M = O(2^k\frac{n^2}{k})$ [Ka2], we conclude that the total time complexity of the algorithm is $\Theta(\min(Mnk + kmn\min(k,\sqrt{n}),Mn + k^2n^3) = O(2^kn^3)$. Note that for finding all minimum size *edge separators* we need to find all minimum separating edge sets between at most $n-1$ pairs of vertices [BaPr]. In contrast our algorithm needs to consider kn pairs in order to find all minimum size separating vertex sets.

5. Parallel algorithm

In this section we present a parallel algorithm for finding all minimum size separating vertex sets of G. Note that if k is bigger than *poly*$\log(n)$ than the time complexity of the sequential algorithm from the previous section might be greater than polynomial in n. The parallel algorithm is very similar to the sequential one, but every step of it will use a parallel version.

Algorithm

1. Find connectivity k of a graph G
2. a). Take a set K of k vertices of G. Check if the set K is a separating k-set of G.

   b). Form all pairs of vertices (x,v), where $x \in K$ and $v \in G$. There are kn pairs. Number these pairs.
3. For every pair (x,v) make a copy of the graph G and add an edge (y,z) for every pair (y,z) whose number is smaller than the number of the pair (x,v). Call this graph G_{xv}.
4. For every pair (x,v) create a directed graph $\overline{G}_{xv}$ by using the Even-Tarjan reduction. Find the maximum flow f_{xv} from x'' to v' in $\overline{G}_{xv}$.
5. If $f_{xv} = k$ do
6. Shrink all strongly connected components of the residual graph of $\overline{G}_{xv}$ with respect to f_{xv}. The resulting acyclic directed graph is L_{xv}.

7. Find all closed sets of L_{xv}.

Now, we will show how to implement each step efficiently in parallel. For step 1 we will use idea from the sequential algorithm. We will take k vertices of G (K) and find the maximum flow between every vertex K and every vertex of G. Note that since we can run all of these kn maximum flows in parallel, we can stop as soon as we find the maximum flow for one of the pairs.

For maximum flow we can use several different algorithms. The first algorithm is deterministic and is better for small values of k. It uses the straight forward implementation of the above algorithm. It takes $O(k\log n)$ parallel time using $O(kn\frac{N^\alpha}{\log}n)$ EREW PRAM processors, where N^α is the number of processors needed for matrix multiplication [KarRa]. We use matrix multiplication for finding the shortest path in $\overline{G}_{xv}$ for each pair (x,v) in parallel. We need to repeat this at most k times.

The second algorithm is a randomized algorithm, but runs faster for large k. We find a maximum flow for every pair in a unit network using randomized parallel algorithm for matching [MuVaVa]. That takes $O(\log^2 n)$ parallel time using $O(kn^2N^\alpha)$ CRCW PRAM processors and gives an RNC algorithm.

First part of Step 2 takes $O(\log n)$ parallel time using $O(n+m)$ CRCW PRAM processors [ShVi]. Second part of Step 2 takes $O(\log nk/\log\log nk)$ parallel time using $O(nk/\log kn)$ CREW PRAM processors using parallel prefix [KarRa]. Step 3 takes $O(1)$ parallel time using $O(nk(m+n))$ EREW PRAM processors. Step 4 is essentially the same as step 1. Step 6 takes $O(\log n)$ parallel time with $O(knN^\alpha)$ CRCW PRAM processors [Hi]. Step 7 takes $O(\log n)$ parallel time using $O(M_{xv}^2n^2)$ CRCW PRAM processors. We will show the implementation of this step below.

Let L_{xv} be the residual graph of $\overline{G}$ with respect to a maximum flow from x'' to v' with shrunk strongly connected components. Recall that there is one-to-one correspondence between the k-sets separating x and v in G and the *antichains* in L_{xv}. If we add to L_{xv} all edges between every vertex y and all of its successors then we get a transitive closure L_{xv}^+. Then every antichain in L_{xv}^+ still gives an $(S,\overline{S})$ cut in N_{xv}. The network L_{xv}^+ is still acyclic and directed. We will use the adjacency matrix of L_{xv}^+ to if two vertices are incomparable.

For the problem of finding all antichains in a transitive closure of an acyclic network we will use well-known doubling technique. We will first find all antichains of sizes of powers of 2, and then use them to find all other

antichains of all other sizes. Take every single vertex as an antichain. Take all antichains of the current size and take all possible unions between them. Now, check all created sets and remove all sets which are not antichains of the double size or repetitions. Repeat that procedure log$n$ times. This creates all antichains of the sizes of powers of 2. Now we can use these antichains and get antichains of all others sizes using at most log$n$ antichains of the sizes powers of 2.

Algorithm

1. Form a transitive closure L^+ of an input network L.
2. Take every vertex as antichain
3. Repeat log$n$ times

   Find all antichains of the double size using antichains of the current size
4. Find all other antichains of all other sizes using at most log$n$ independent sets of sizes of powers of 2.
5. Find all closed sets in the network using antichains.

Let us state the time complexities and processor bounds for each step of the above algorithm. We establish the bounds for Steps 3 and 4 below. Step 1 of the above algorithm runs in $O(\log n)$ time using $O(N^{\alpha})$ EREW PRAM processors, where N^{α} is the number of processors used for matrix multiplication [KarRa]. Step 2 runs in $O(1)$ parallel time using $O(n)$ EREW PRAM processors. Step 3 runs in $O(\log n)$ parallel time using $O(M_{st}^2 n^2)$ CRCW PRAM processors as shown below. Step 4 runs in $O(\log n)$ parallel time using $O(M_{st}^2 n^2)$ CRCW PRAM processors [TaVi]. Step 5 runs in $O(1)$ parallel time using $O(M_{st} m)$ CREW PRAM processors. Hence, total parallel time spent is $O(\log n)$ using $\Theta(M_{st}^2 n^2) = O(4^k \frac{n^6}{k^2})$ CRCW PRAM processors.

Let us see in detail the implementation of Step 3. Note that the number of antichains in L_{st}^+ is equal to the number of k-sets which separates s and t in $G(M_{st})$. First of all, we take all S_1 and S_2 which are two different antichains of current size i, to create at most M_{st}^2 sets of size $2i$. In order to check if a created set is an antichain of size $2i$ we need to check two properties. First, that the created set is an antichain, and second that its cardinality of a set is $2i$. For the first property we will check the $n \times n$ adjacency matrix of L_{xv}^+. For the second property we check if $a \,!= b$ for every pair of elements (a,b), where $a \in S_1$ and $b \in S_2$. So we can check each set in $O(1)$ parallel time using

$O(n^2)$ CRCW PRAM processors. Hence, Step 3 runs in $O(\log n)$ parallel time using $O(M_{st}^2 n^2)$ CRCW PRAM processors and Step 4 runs in $O(\log n)$ parallel time using $O(M_{st}^2 n^2)$ CRCW PRAM processors.

We have to run the above algorithm for kn L_{xv} graphs, one for each pair (x,v). But

$$\sum_{i=1}^{i=k}\sum_{j=1}^{j=n} M_{ij}^2 n^2 \le (\sum_{i=1}^{i=k}\sum_{j=1}^{j=n} M_{ij})^2 n^2 = M^2 n^2 \le 4^k \frac{n^6}{k^2},$$

since no separating k-set is created twice.

Hence, step 7 of the parallel algorithm for finding all minimum size separating vertex sets runs in $O(\log n)$ times using $O(4^k \frac{n^6}{k^2})$ CRCW PRAM processers for all kn pairs of vertices (x,v) $x \in K$ and $v \in V$.

The entire parallel algorithm runs in $O(k \log n)$ deterministic time using $\Theta(M^2 n^2 + knN^\alpha) = O(4^k \frac{n^6}{k^2} + knN^\alpha)$ CRCW PRAM processors, or runs in $O(\log^2 n)$ randomized parallel time using $\Theta(M^2 n^2 + kn^2 N^\alpha) = O(4^k \frac{n^6}{k^2} + kn^2 N^\alpha)$ CRCW PRAM processors, where N^α is the number of processors needed for matrix multiplication.

Acknowledgement

I would like to thank Vijaya Ramachandran for introducing me to this problem and many crucial discussions and suggestions. I also wish to thank Roberto Tamassia for valuable discussions and comments. I wish to thank Dan Gusfield for pointing out application of this result for network reliability and for suggesting several references for it.

REFERENCES

[Ba] M. O. Ball, "Computing network reliability," *Operations Research*, vol. 27, No. 4, July-August, 1979, pp. 823-838.

[BaPr] M. O. Ball, J. S. Provan, "Calculating bounds on reachability and connectedness in stochastic networks," *Networks*, vol. 13, 1983, pp. 253-278.

[BeX] M. Becker et al., "A probabilistic algorithm for vertex connectivity of graphs," *Inform. Proc. Lett.*, vol. 15, no. 3, 1982, pp. 135-136.

[Bu] J. A. Buzacott, "A recursive algorithm for finding reliability measures related to the connection of nodes in a graph," *Networks*, vol. 10, 1980, pp. 311-327.

[Co] C. J. Colbourn, "The reliability polynomial," *ARS Combinatorica*, 21-A, 1986, pp. 31-58.

[Co] R. Cole, "Parallel merge sort," *Proc. 27th IEEE Ann. Symp. on Foundations of Computer Science*, 1986, pp. 511-516.

[Ev1] S. Even, *Graph Algorithms*, Computer Science Press, Rockville, MD, 1979.

[Ev2] S. Even, "An algorithm for determining whether the connectivity of a graph is at least *k*," *SIAM J. Comput.*, vol. 4, 1975, pp. 393-396.

[EvTa] S. Even, R. E. Tarjan, "Network flow and testing graph connectivity," *SIAM J. Comput.*, vol. 4, 1975, pp. 507-518.

[Ga] Z. Galil, "Finding the vertex connectivity of graphs," *SIAM J. Comput.*, vol. 9, 1980, pp. 197-199.

[GiSo] M. Girkar, M. Sohoni, "On finding the vertex connectivity of graphs," *Tech. Report ACT-77*, Coordinated Science Laboratory, University of Illinois, Urbana, Il, May, 1987.

[Hi] D. S. Hirshberg, "Parallel algorithms for transitive closure and the connected components problems," *Proc. 8th Ann. ACM Symp. on Theory of Computing* New York, 1976, pp. 55-57.

[HiChSa] D. S. Hirshberg, A. K. Chandra, D. V. Sarwate, " Computing connected components on parallel computers," *Communications of ACM* vol. 22, 1979, pp. 461-464.

[HoTa] J. E. Hopcroft, R. E. Tarjan, "Dividing a graph into triconnected components," *SIAM J. Comput.*, 1973, pp. 135-158.

[Ka1] A. Kanevsky, "On the number of minimum size separating vertex sets in a graph," *Tech. Report ACT-80*, Coordinated Science Laboratory, University of Illinois, Urbana, Il, July, 1987.

[Ka2] A. Kanevsky, "Compact Representation of the separating *k*-sets of a graph," *Tech. Report ACT-88*, Coordinated Science Laboratory, University of Illinois, Urbana, Il, January, 1988.

[KanRa1] A. Kanevsky, V. Ramachandran, "Improved algorithms for graph four-connectivity," *Proc. 28th IEEE Ann. Symp. on Foundation of Computer Science*, Oct 1987, pp. 252-259.

[KanRa2] A. Kanevsky, V. Ramachandran, "A characterization of separating pairs and triplets in a graph," *Tech. Report ACT-79*, Coordinated Science Laboratory, University of Illinois, Urbana, Il, July, 1987.

[KarRa] R. M. Karp, V. Ramachandran, "A survey of parallel algorithms for shared memory machines," *Handbook of Theoretical Computer Science*, North Holland, 1988, to appear.

[LiLoWi] N. Linial, L. Lovasz, A. Wigderson, "A physical interpretation of graph connectivity, and its algorithmic applications," *Proc. 27th IEEE Ann. Symp. on Foundations of Computer Science*, 1986.

[Lo] L. Lovasz,"Computing ears and branchings in parallel," *Proc. 26th IEEE Ann. Symp. on Foundations of Computer Science*, 1985, pp. 464-467.

[MaScVi] Y. Maon, B. Schieber, U. Vishkin, "Parallel ear decomposition search (EDS) and st-numbering in graphs," *VLSI Algorithms and Architectures*, Lecture Notes in Computer Science vol. 227, 1986, pp. 34-45.

[Me] K. Menger, "Zur Allgemeinen Kurventheorie," *Fund. Math.* 10, 1927, pp. 96-115.

[MiRa1] G. L. Miller, V. Ramachandran, "Efficient parallel ear decomposition with applications," unpublished manuscript, MSRI, Berkeley, CA, January 1986.

[MiRa2] G. L. Miller, V. Ramachandran, "A new graph triconnectivity algorithm and its parallelization," *Proc. 19th ACM Ann. Symp. on Theory of Computing*, New York, NY, 1987, pp. 335-344.

[MuVaVa] K. Mulmuley, U. V. Vazirani, V. V. Vazirani, "Matching is as easy as matrix inversion," *Combinatorica*, 7, 1, pp. 105-114.

[PiQu] J. C. Picard, M. Queyranne, "On the structure of all minimum cuts in a network and applications," *Mathematical Programming Study* 13, 1980, pp. 8-16.

[RaVi] V. Ramachandran, U. Vishkin, "Efficient parallel triconnectivity in logarithmic time," *Aegean Workshop on Computing*, to appear, 1988.

[RaCo] A. Ramanathan, C. J. Colbourn, "Counting almost minimum cutsets with reliability applications," *Mathematical Programming*, 39, 1987, 253-261.

[Ro] A. Rosenthal, "Computing the reliability of complex networks," *SIAM J. Appl. Math.*, vol. 32, No. 2, March 1977, pp. 384-393.

[ScBa] L. Schrage, K. R. Baker, "Dynamic programming solution of sequencing problems with precedence constraints," *Operations Research*, vol. 26, No. 3, 1978, pp. 444-449.

[ShVi] Y. Shiloach, U. Vishkin, "An $O(\log n)$ parallel connectivity algorithm," *J. Algorithms*, vol. 3, 1982, pp. 57-63.

[Ta] R. E. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, 1972, pp. 146-160.

[TaVi] R. E. Tarjan, U. Vishkin, "An efficient parallel biconnectivity algorithm," *SIAM J. Comput.*, 14, 1985, pp. 862-874.

[Tu] W. T. Tutte, *Connectivity in Graphs*, University of Toronto Press, 1966.

[Wh] H. Whitney, "Non-separable and planar graphs," *Trans. Amer. Math. Soc.* 34, 1932, pp. 339-362.