Fast Parallel Algorithms for Testing k-connectivity of Directed and Undirected Graphs *

Weifa Liang, Brendan D. McKay
Department of Computer Science
The Australian National University
Canberra, ACT 0200, Australia
{wliang, bdm}@cs.anu.edu.au

Abstract

It appears that no NC algorithms have previously appeared for testing a directed graph for k-edge connectivity or k-vertex connectivity, even for fixed $k > 1$. Using an elementary flow method we give such algorithms, with time complexity $O(k \log n)$ using $nP(n,m)$ or $(n+k^2)P(n,m)$ processors, respectively. Here, n is the number of vertices, m is the number of edges, $P(n,m)$ is the number of processors needed to find some path in time $O(\log n)$ time between two specified vertices in a directed graph with $O(n)$ vertices and $O(m)$ edges, and the computation model is a CRCW PRAM. These algorithms of course apply also to undirected graphs, but using sparse certificates we can improve the factors $P(n,m)$ to $P(n,kn)$ for both types of connectivity. This is better in time by a factor of $O(k)$ over previous algorithms for undirected graphs. We also note that edge connectivity is NC-reducible to vertex connectivity even if k is not fixed.

Keywords: k-edge connectivity, k-vertex connectivity, parallel algorithms, disjoint-paths, graph problems.

1 Introduction

Connectivity of graphs (vertex connectivity as well as edge connectivity) is considered to be one of the classic subjects in graph theory, and has many practical applications, e.g., in reliability of communication networks, chip and circuit design, and cluster analysis. Designing efficient parallel algorithms for testing graph connectivity is clearly a basic problem in parallel computation. In this paper we present parallel algorithms for vertex and edge connectivity for each of directed and undirected graphs. In all four cases our algorithms have better theoretical complexity than previously published algorithms.

The model of parallel computation used here is that of a concurrent read and concurrent write parallel random access machine (CRCW PRAM). In this model, simultaneous access by more than one processor to the same memory location for both read and write is allowed. In the event that several processors attempt to write to the same memory location simultaneously, an arbitrary one succeeds.

We will write our complexity measures in terms of three basic quantities. $P(n,m)$ is the number of processors needed to find a directed path in time $O(\log n)$ between two specified vertices in a directed graph with $O(n)$ vertices and $O(m)$ edges. Secondly, $T(n,m)$ is the number of processors needed to determine the set of vertices reachable from a specified vertex in a directed graph with $O(n)$ vertices and $O(m)$ edges, in time $O(\log n)$. The current best results for $P(n,m)$ and $T(n,m)$ are both $n^{2.376}$, using matrix multiplication [3]. Finally, $C(n,m)$ is the number of processors needed to find the connected components of an undirected graph with n vertices and m edges in time $O(\log n)$. In this case the current best result is $C(n,m) = (n+m)\alpha(n,m)/\log n$, where $\alpha(n,m)$ is a functional inverse of Ackermann's function [2].

First consider the case of directed graphs. To our knowledge, there are no published NC algorithms for k-vertex connectivity or k-edge connectivity for any k except $k = 1$. (An NC algorithm is one which takes time $O(\log^c n)$ for some constant c using a polynomial number of processors.) We give algorithms taking time $O(k \log n)$, using $nP(n,m)$ processors in the case of edge connectivity, and $(n+k^2)P(n,m)$ processors in the case of vertex connectivity. These algorithms use simple

This work was partially supported by ACSys grant.

0-7803-2018-2/95/$4.00

implementations of the Ford-Fulkerson network flow algorithm. In the case where the connectivity is less than k, we also find separating sets within the same time bound. The best deterministic sequential algorithm for the edge-connectivity is due to Gabow [6] and takes time $O(\lambda m \log(n^2/m))$ using a matroid approach. Here, λ is the value of the edge connectivity.

Next consider the case of undirected graphs. Khuller and Schieber [7] present algorithms for k-edge connectivity and k-vertex connectivity using time $O(k^2 \log n)$. The required number of processors is $nkC(n,m)$ and $(nk+k^3)C(n,m)$, respectively. The number of processors needed in the latter case was recently reduced to $(nk+k^3)C(n,nk)$ by Cheriyan, Kao and Thurimella [1], using a sparse certificate technique. Our algorithms improve the time requirement to $O(k \log n)$ at some expense to the number of processors. Clearly, if an undirected edge is considered to be a pair of oppositely directed edges, our algorithms for directed graphs can be used with the same complexity. By using sparse certificates we can reduce the number of processors to $nP(n,nk)$ for edge connectivity and $(n+k^2)P(n,nk)$ for vertex connectivity. The best deterministic sequential algorithm for vertex connectivity takes time $O(nm)$ for fixed k (Even [4]). For edge connectivity, the above mentioned algorithm of Gabow can be used, but for some values of the parameters one of the two algorithms of Matula [8] is better, as they use time $O(nm)$ and $O(\lambda n^2)$ respectively.

We also show that the k-edge connectivity problem is NC-reducible to the k-vertex connectivity problem for both undirected graphs and directed graphs, with k arbitrary. Therefore, if the solution for the k-vertex connectivity problem is in NC, then the k-edge connectivity problem is also in NC.

The paper is organized as follows. In Section 2 we give our algorithms for directed graphs, and in Section 3 we give our improvements for undirected graphs. In Section 4 we discuss the NC-reduction of edge connectivity to vertex connectivity.

2 Directed graphs

Let $G(V,E)$ be a directed graph with $|V|=n$ and $|E|=m$. Multiple edges will not be allowed, but could be incorporated without much effort. We will use (i,j) to represent a directed edge from i to j.

Let s and t be distinct vertices in G that are not connected by an edge. An *$s-t$ vertex separator* is a subset $S \subseteq V-\{s,t\}$, such that every path from s to t contains at least one vertex from S. Define $N(s,t)$ to be the minimum cardinality of an $s-t$ vertex separator. By Menger's Theorem, $N(s,t)$ is also the maximum number of vertex disjoint paths from s to t. The *vertex connectivity* $\kappa(G)$ of G is defined to be $\kappa(G) = \min\{N(s,t) \mid s,t \in V, s \neq t, (s,t) \notin E\}$.

Similarly, let s and t be distinct vertices of G and define an *$s-t$ edge separator* to be a set $Q \subseteq E$ such that every path from s to t uses at least one edge in Q. Define $M(s,t)$ to be the minimum cardinality of an $s-t$ edge separator. By Menger's Theorem, $M(s,t)$ is also the maximum number of edge disjoint paths from s to t. The *edge connectivity* of G is defined to be $\lambda(G) = \min\{M(s,t) \mid s,t \in V, s \neq t\}$.

Our fundamental approach will be the use of network flows to find disjoint paths. Consider the directed graph $G(V,E)$ to be a 0-1 network in which the capacity of each edge is one. Suppose G carries some 0-1 legal $s-t$ flow f. Define the auxiliary directed graph $\tilde{G} = G(V,\tilde{E})$ as follows: For each distinct $u,v \in V$, $(u,v) \in \tilde{E}$ if and only if either $(u,v) \in E$ and $f(u,v)=0$, or $(v,u) \in E$ and $f(v,u)=1$. Note that $\tilde{G}$ is similar to the "residue network" of G and f, but has only edges of capacity one. It is still true that paths in $\tilde{G}$ are augmenting paths in G.

Theorem 2.1. The flow f is a maximum $s-t$ flow in G if and only if $\tilde{G}$ has no directed $s-t$ paths.
Proof. See Chapter 6 in [5]. □

This theorem implies an algorithm for finding an $s-t$ flow of total value k, or proving there is none. The details are as follows.

Lemma 2.2. Given a 0-1 network $G(V,E)$ and $s,t \in V$, testing whether the value of the flow from s to t is no less than k can be done in $O(k \log n)$ time with $P(n,m)$ processors.
Proof. The construction of $\tilde{G}$ can be done in $O(\log n)$ time using $(m+n)/\log n$ processors. Finding an $s-t$ path requires $O(\log n)$ time and $P(n,m)$ processors. The updating of P can be finished in $O(\log n)$ time using $O(kn)$ processors, as follows: firstly we sort the edges in P by their key (u,v), which costs $O(\log n)$ time and $O(kn)$ processors because $|P| \leq kn$. Then for each edge $(u,v) \in P'$, we look up the sorted list on P by binary searching to see whether (v,u) is in P. If it does, delete edges (u,v) and (v,u) from these two lists, respectively. This step requires $O(\log n)$

Procedure *Flow*(G,k,i,s,t);
/* P is the set of edges with flow 1; and i is actual flow value */
Initialize $P := \emptyset$;
for $i := 0$ **to** k **do**
Form $\tilde{G} = G(V,\tilde{E})$, where $\tilde{E} := (E-P) \cup \{(u,v) \mid (v,u) \in P\}$;
If there is no directed s–t path P' in $\tilde{G}$, **return** FAIL;
Else $P := P \cup P' - \{(u,v),(v,u) \mid (u,v) \in P \text{ and } (v,u) \in P'\}$
endfor
return P.

time and $O(n)$ processors because $|P'| \leq n$. The remaining elements in these two lists are merged into a new list P. □

Lemma 2.3. If algorithm *Flow* fails because the maximum flow is $k' < k$, an edge separator of size k' can be found using an additional amount of $O(\log n)$ time and $\max\{m, T(n,m)\}$ processors.
Proof. Consider the final value of $\tilde{G}$ constructed by the algorithm. In time $O(\log n)$ using $T(n,m)$ processors, compute the set W of vertices reachable from s in $\tilde{G}$. Then, by standard flow theory, $Q = \{(u,v) \in E \mid u \in W, v \notin W\}$ is an edge separator of size k'. □

Theorem 2.4. There is an algorithm to test whether a directed graph is k-edge connected, and if not to find an edge separator of size less than k. The running time is $O(k \log n)$ and the number of processors is $nP(n,m)$.
Proof. Let $V = \{v_1, v_2, \ldots, v_n\}$. By a theorem of Schnorr [10], the edge connectivity of G is $\lambda(G) = \min\{M(v_i, v_{i+1}) \mid 1 \leq i \leq n\}$, where $v_{n+1} = v_1$. We can use procedure *Flow* to test in parallel if the n associated flow problems all have solution at least k. If one does not, we can find an edge separator as described in Lemma 2.3. Obviously $\max\{m, T(n,m)\} = O(nP(n,m))$, so $nP(n,m)$ processors will suffice. □

We now consider vertex connectivity for directed graphs. The following lemma was inspired by a similar lemma of Even [4] for undirected graphs.

Lemma 2.5. Suppose $V = \{v_1, v_2, \ldots, v_n\}$. Define two auxiliary directed graphs $G' = G(V', E')$ and $G'' = G(V'', E'')$ as follows. $V' = V'' = V \cup \{z\}$; $E' = E \cup \{(z, v_i) \mid 1 \leq i \leq k\}$; $E'' = E \cup \{(v_i, z) \mid 1 \leq i \leq k\}$. Then $\kappa(G) \geq k$ if and only if $N(v_i, v_j) \geq k$ in G for $1 \leq i \neq j \leq k$, $N(z, v_j) \geq k$ in G' for $k+1 \leq j \leq n$, and $N(v_j, z) \geq k$ in G'' for $k+1 \leq j \leq n$.
Proof. If $\kappa(G) < k$, then we can partition V into non-empty subsets $V = S \cup W \cup T$ such that $|W| < k$ and W is an s–t vertex separator for any $s \in S$ and $t \in T$. If $V_k = \{v_1, v_2, \ldots, v_k\}$ intersects both S and T, we have $N(s,t) < k$ in G for any $s \in S \cap V_k$, $t \in T \cap V_k$. If $V_k \subseteq S \cup W$, $N(z,t) < k$ in G' for any $t \in T - V_k$. If $V_k \subseteq W \cup T$, $N(s,z) < k$ in G'' for any $s \in S - V_k$. □.

In order to apply Lemma 2.5 we use the standard method for finding vertex-disjoint paths with the help of flows. Define the auxiliary directed graph $\overline{G} = G(\overline{V}, \overline{E})$, where $\overline{V} = \{v', v'' \mid v \in E\}$, and $\overline{E} = \{(v', v'') \mid v \in V\} \cup \{(u'', v'), (v'', u') \mid (u,v) \in E\}$.

Lemma 2.6. For any distinct vertices $s, t \in G$, the maximum number of vertex-disjoint s–t paths in G equals the maximum number of edge-disjoint s''–t' paths in $\overline{G}$.
Proof. See Chapters 5 and 6 in [5]. □

Theorem 2.7. There is an algorithm to test whether a directed graph is k-vertex connected, and if not to find a vertex separator of size less than k. The running time is $O(k \log n)$ and the number of processors is $(n + k^2)P(n,m)$.
Proof. The $k(k-1) + 2(n-k)$ disjoint path problems defined in Lemma 2.5 can be solved in parallel using Lemma 2.6. We can form the three networks in $O(1)$ time using $m + n$ processors and solve the flow problems in $O(k \log n)$ time using $P(n,m)$ processors each, with procedure *Flow*. Since obviously $m + n = O(nP(n,m))$, the total number of processors required is $(n + k^2)P(n,m)$.

If all the flows have value at least k, we know that G is k-vertex connected. Otherwise it is easy to find a small vertex separator. Suppose $G^* = G(V^*, E^*)$ is the network with $N(s'', t') < k$. Let W^* be the set of vertices reachable from s'' in $G(V^*, E_1^* \cup \tilde{E})$, where E_1^* is the subset of

E^* consisting of edges of the form (u'', v'), and $G(V^*, \tilde{E})$ is the final graph $\tilde{G}$ made by procedure *Flow*. Then the set of all vertices $v \in V$ such that $v' \in W^*$ and $v'' \notin W^*$ is an s–t vertex separator in G with size less than k. Finding it with the same number of processors in $O(\log n)$ time is easy. □

3 Undirected graphs

An undirected graph can be considered as a directed graph in which the edges come in pairs $(u, v), (v, u)$. Under that interpretation it is easy to see that the definitions of connectivity and separators given in the previous section correspond to the normal definitions given for undirected graphs. Consequentially, Theorems 2.4 and 2.7 apply equally to undirected graphs. However, we can reduce the required number of processors by using "sparse certificates".

Throughout this section, $G(V, E)$ is an undirected graph with $|V| = n$ and $|E| = m$. A *sparse certificate for k-vertex connectivity* is a subgraph H of G that has $O(kn)$ edges and such that every vertex separator of size less than k in H is also a vertex separator in G. A *sparse certificate for k-edge connectivity* is defined similarly.

Now we recall a search technique on undirected graphs called *scan-first search*, due to Cheriyan, Kao and Thurimella [1]. Vertices are initially "unmarked". Then the following rules are applied repeatedly until all vertices are marked:
(i) If there is a marked vertex v with at least one unmarked neighbour, mark all the unmarked neighbours w of v;
(ii) If there is no such v, mark an unmarked vertex.
The edges $\{v, w\}$ encountered in step (i) form a maximal spanning forest of G.

Define $E_0 = E$, and let E_i be the edges of a scan-first spanning forest of $G(V, E - E_1 - \cdots - E_{i-1})$ for $i = 1, \ldots, k$. Then define $G_k = G(V, E_1 \cup E_2 \cup \cdots \cup E_k)$.

Lemma 3.1. G_k is a sparse certificate for both k-vertex connectivity and k-edge connectivity.
Proof. For edge connectivity, this was proved in [7] and [9]; in fact we can use any maximal spanning forests, not necessarily scan-first. For vertex connectivity, this lemma was proved in [1].

Lemma 3.2. [1]. A scan-first search spanning forest of G can be found in $O(\log n)$ time using $C(n, m)$ processors.

Therefore, we have

Theorem 3.3. Given an undirected graph $G(V, E)$, a k-vertex sparse certificate and k-edge sparse certificate can be found in $O(k \log n)$ time using $C(n, m)$ processors.
Proof. This follows immediately from the preceding two lemmas. □

Theorem 3.4. There is an algorithm to test whether an undirected graph is k-edge connected, and if not to find an edge separator of size less than k. The running time is $O(k \log n)$ and the number of processors is $nP(n, nk)$.
Proof. Find a sparse certificate for k-edge connectivity, and test it as in Theorem 2.4. The required number of processors is $\max\{C(n, m), nP(n, nk)\}$. However, as indicated in the introduction, $C(n, m) = O((n + m)\alpha(n, m)/\log n)$ and also clearly $P(n, nk) = \Omega(nk/\log n)$. Hence the term $nP(n, nk)$ dominates. □

Theorem 3.5. There is an algorithm to test whether an undirected graph is k-vertex connected, and if not to find a vertex separator of size less than k. The running time is $O(k \log n)$ and the number of processors is $(n + k^2)P(n, nk)$.
Proof. Use the same approach as for Theorem 3.4. □

4 *NC* reduction of edge connectivity to vertex connectivity

Given the results in the previous sections, we know that k-edge and k-vertex connectivity are in *NC* for $k = O(\log^c n)$, where c is any constant. The situation for arbitrary $k = k(n)$ remains unsolved. In this section we note that, in order to prove that k-edge connectivity is in *NC*, it would suffice to prove that k-vertex connectivity is in *NC*, both for directed and for undirected graphs.

Theorem 4.1. Let G be an undirected graph, and let $L(G)$ be the line-graph of G. Then $\lambda(G) = \min\{\delta(G), \kappa(L(G))\}$ where $\delta(G)$ is the minimum degree of G.
Proof. By the definition of $L(G)$, a k-vertex separator of $L(G)$ is a k-edge separator of G. Conversely, a minimum k-edge separator of G is either a k-vertex separator of $L(G)$ or corresponds to the edges incident with a single vertex of G. □

Essentially the same method works for directed graphs, if we are careful to use the correct line-graph. For a directed graph $G(V,E)$ define $L(G)$ to be the directed graph with vertex set E, with an edge from (u,v) to (u',v') exactly when $v = u'$.

Theorem 4.2. Let G be a directed graph, and let $L(G)$ be the line-graph of G. Then $\lambda(G) = \min\{\delta^+(G), \delta^-(G), \kappa(L(G))\}$ where $\delta^+(G)$ and $\delta^-(G)$ are the minimum in-degree and minimum out-degree of G, respectively.

Proof. A k-vertex separator of $L(G)$ is a k-edge separator of G. Conversely, a minimum k-edge separator of G is either a k-vertex separator of $L(G)$ or corresponds to all the edges entering a single vertex of G or all the edges leaving a single vertex of G. □

In both the undirected and directed cases, the construction of $L(G)$ can be achieved in time $O(\log n)$ using $n + m^2$ processors.

References

[1] J. Cheriyan, M-Y Kao and R. Thurimella, Scan-first search and sparse certificates: an improved parallel algorithm for k-vertex connectivity, *SIAM J. Comput.*, Vol. 22, No. 1, 1993, pp. 157-174.

[2] R. Cole and U. Vishkin, Approximate and exact parallel scheduling with applications to list, tree and graph problems, *Proc. 27th Annual IEEE Symp. of Foundations of Computer Science*, 1986, pp. 478-491.

[3] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions, *Proc. 19th Annual ACM Symp. on Theory Computing*, 1987, pp. 1-6.

[4] S. Even, An algorithm for determining whether the connectivity of a graph is at least k, *SIAM J. Comput.*, Vol. 4, No. 3, 1975, pp.393-395.

[5] S. Even, *Graph Algorithms*, Computer Science Press, New York, 1979.

[6] H. N. Gabow, A matroid approach to finding edge connectivity and packing arborescences, *Proc. 23rd Annual ACM Symp. on Theory of Computing*, 1991, pp. 112-122.

[7] S. Khuller and B. Schieber, Efficient parallel algorithms for testing connectivity and finding disjoint $s-t$ paths in graphs, *SIAM J. Comput.*, Vol. 20, No. 2, 1991, pp.352-375.

[8] D. W. Matula, Determining edge connectivity in $O(nm)$, *Proc. 28th Annual Symp. of Foundations of Computer Science*, 1987, pp.249-251.

[9] H. Nagamochi and T. Ibaraki, Linear time algorithm for finding k-edge-connected and k-vertex-connected spanning subgraphs, *Algorithmica*, Vol. 7, 1992, pp. 583-596.

[10] C. P. Schnorr, Bottlenecks and edge connectivity in unsymmetrical networks, *SIAM J. Comput.*, Vol. 8, No. 2, 1979, 265-274.