Parallel Recognition of Series-Parallel Graphs

David Eppstein
Department of Information and Computer Science
University of California, Irvine, CA 92717

October 4, 1990

Abstract

Recently, He and Yesha gave an algorithm for recognizing directed series parallel graphs, in time $O(\log^2 n)$ with linearly many EREW processors. We give a new algorithm for this problem, based on a structural characterization of series parallel graphs in terms of their ear decompositions. Our algorithm can recognize undirected as well as directed series parallel graphs. It can be implemented in the CRCW model of parallel computation to take time $O(\log n)$. In the EREW model the time is $O(\log^2 n)$ but the number of processors required improves the bounds of the previous algorithm.

1 Introduction

A directed graph G is *two-terminal series parallel*, with terminals s and t, if it can be produced by a sequence of the following operations:

1. Create a new graph, consisting of a single edge directed from s to t.
2. Given two two-terminal series parallel graphs X and Y, with terminals s_X, t_X, s_Y, and t_Y, form a new graph $G = P(X, Y)$ by identifying $s = s_X = s_Y$ and $t = t_X = t_Y$. This is known as the *parallel composition* of X and Y.
3. Given two two-terminal series parallel graphs X and Y, with terminals s_X, t_X, s_Y, and t_Y, form a new graph $G = S(X, Y)$ by identifying $s = s_X$, $t_X = s_Y$, and $t = t_Y$. This is known as the *series composition* of X and Y.

An undirected graph is two terminal series parallel with terminals s and t if for some orientation of its edges it forms a directed two terminal series parallel graph with those terminals. A directed or undirected graph is *series parallel* if for some two vertices s and t it is two terminal series parallel with those terminals.

Recognition of series parallel graphs is one of the classical problems in the design of algorithms, and it is well known that this can be performed in linear time [14]. Further, given a decomposition of a series parallel graph according to the above operations, one can perform many other computations on the graph in linear time; these computations include problems such as maximum matchings, maximum independent sets, minimum dominating sets, and other problems including many that for general graphs are NP-complete [3, 9, 12]. Such a decomposition can be constructed in linear time.

Recently, a parallel algorithm was given by He and Yesha for recognizing directed series parallel graphs and providing a decomposition into the series parallel composition operations [7]. This algorithm takes time $O(\log^2 n)$, and uses $O(n+m)$ shared-memory parallel processors. Two processors never attempt to access the same memory cell at the same time; this model of parallel computation is known as EREW (for Exclusive Read Exclusive Write).

In this paper we present a new algorithm for both directed and undirected series parallel graph recognition and decomposition, which takes time $O(\log n)$ with $C(m,n)$ processors; here $C(m,n)$ is the number of processors required to compute connected components of a graph in logarithmic time. The best bound known for this is $C(m,n) = O(m\alpha(m,n)/\log n)$ [5]. We use the stronger concurrent read concurrent write (CRCW) model of parallelism; however any CRCW algorithm can be executed on an EREW machine with a logarithmic loss of time and efficiency. In our case this results in time bounds of $O(\log^2 n)$, matching the previous result; however the number of processors we require, $C(n,m)$, is an improvement over the previous $O(m+n)$.

Our algorithm is based on the concept of an *open ear decomposition* of a graph. In the next section we define this concept, and show that undirected two terminal series parallel graphs may be characterized by a *nesting* property of their ear decompositions. Next we show how to reduce the problem of directed series parallel graph recognition and the cases in which terminals have not been specified to the undirected two terminal case. In the following section we describe how to test the nesting property in parallel, and in the final section we describe how to combine all these steps to give a series parallel graph recognition algorithm.

2 Nested Ear Decompositions

An *ear decomposition* of an undirected graph G is defined to be a partition of the edges of G into a sequence of *ears* $E_1, E_2, \ldots E_n$. Each ear is a path in the graph with the following properties:

1. If two vertices in the path are the same, they must be the two endpoints of the path.
2. The two endpoints of each ear E_i, $i > 1$, appear in previous ears E_j and E_j', with $j < i$ and $j' < i$.
3. No interior point of E_i is in E_j for any $j < i$.

Typically there are further restrictions on the first ear E_1; for instance E_1 may be required to be a single vertex or edge.

An *open ear decomposition* is one in which even the two endpoints of each ear must be distinct; i.e., each path must be simple. We say that an ear decomposition or open ear decomposition *starts from* a certain path P if $E_1 = P$.

Ear decompositions have a number of uses, in particular in computing the connectivity of a graph. For instance, the following theorem is well known.

Lemma 1 *(Whitney [15]) An undirected graph is biconnected (2-vertex-connected) if and only if it has an open ear decomposition starting from a single edge.* $\square$

Given a graph G and an open ear decomposition $ED = \{E_1, E_2, \ldots E_k\}$ of G, we define function $e_{ED}(v)$ to be E_i if v appears as an interior vertex in ear E_i. If v is an endpoint of the initial ear E_1, then let $e_{ED}(v) = E_1$ for completeness. By property 3 of the ear decomposition, $e_{ED}(v)$ is well defined.

Given a graph G and an open ear decomposition $ED = \{E_1, E_2, \ldots E_k\}$ of G, we say that E_i is *nested* in E_j if $j < i$ and the endpoints of E_i both appear in E_j. For such i and j, let the *nest interval* of E_i in E_j be the path in E_j between the two endpoints of E_i.

We say that ED is *nested* if the following conditions hold:

1. For each $i > 1$ there is some $j < i$ such that E_i is nested in E_j.

2. If two ears E_i and $E_{i'}$ are both nested in the same ear E_j, then either the nest interval of E_i contains that of $E_{i'}$, or vice versa, or the two nest intervals are disjoint; i.e., no two nest intervals in each ear E_j cross each other.

Khuller [8] proposed a class of *tree* ear decompositions, which corresponds to only the first condition above. The class of nested ear decompositions is much more restrictive, and in fact we shall see that it is equivalent to the series parallel property.

Lemma 2 *If ED is nested, the nest intervals in any given ear E_i form a tree, in which I is a child of J if I is a maximal subinterval of J, and I is an adjacent sibling to J if they share a parent and there is no interval between them.*

Proof: Obvious from the definition of a nested decomposition. $\Box$

Lemma 3 *If ear decomposition ED is nested, then for each ear E_i, with $i > 1$ and endpoints x and y, E_i is nested in the greater of the two ears $e_{ED}(x)$ and $e_{ED}(y)$.*

Proof: By induction on i. We assume that the lemma holds for all $i' < i$. Then by the definition of nesting there must be some j with E_i nested in E_j. If $j = e_{ED}(x)$ or $j = e_{ED}(y)$, we are done, because since j contains both endpoints it must, by the second property in the definition of ear decompositions, be the greater of $e_{ED}(x)$ and $e_{ED}(y)$. Otherwise, x and y must also be the endpoints of j; but then the lemma follows from the induction hypothesis. $\Box$

Thus we can extend function e_{ED} from vertices to ears: if E_i has endpoints x and y, define $e_{ED}(E_i)$ to be the the greater of the two ears $e_{ED}(x)$ and $e_{ED}(y)$; then lemma 3 implies that E_i is nested in $e_{ED}(E_i)$.

We say that E_i is *properly nested* in E_j if $E_j = e_{ED}(E_i)$, and that E_i is *contained* in E_j when there is a sequence of ears $E_i, E_k, \ldots E_j$ such that each ear is properly nested in the next; that is, containment is the transitive closure of the proper nesting relation. The following lemma demonstrates that only proper nesting need be considered in testing whether an ear decomposition is nested.

Lemma 4 *Let graph G have an open ear decomposition ED, such that:*

1. *ED starts from a single edge.*

2. *For each $i > 1$ there is some $j < i$ such that E_i is nested in E_j.*

3. *If two ears E_i and $E_{i'}$ are both properly nested in the same ear E_j, either the nest interval of E_i contains that of $E_{i'}$, or vice versa, or the two nest intervals are disjoint.*

Then ED is nested.

Proof: If E_i is nested, but not properly nested, in E_j, then the nest interval of E_i is all of E_j, which clearly cannot cross any other nest interval in E_j. □

Conversely, any nested ear decomposition clearly satisfies conditions 2 and 3. This shows that verification of nestedness for ear decompositions starting from a single edge can be performed as two simple steps. First, we check property 2 above, that every ear is nested in some other ear. Next, we check property 3, by collecting the set of ears properly nested in each ear, and checking that there is no pair that cross.

Lemma 5 *Let ED be a nested ear decomposition, and let E_i be an ear with endpoints x and y. Then x and y separate the subgraph induced by the set of ears contained in ear E_i from the rest of the graph.*

Proof: Let E_j be an ear that touches an ear E_k which is contained in E_i. Clearly it must do so only at its endpoints, and therefore it is nested in E_k. If it is properly nested in E_k, it is contained in E_i, and cannot furnish a counterexample to the lemma.

Otherwise, let E_j have the same endpoints as E_k. If $k \neq i$, E_j is again contained in E_i, because it is properly nested in the same ear as E_k. Otherwise, E_j touches the collection of contained ears only at vertices x and y, and is therefore separated from them by those two vertices. □

Lemma 6 *If undirected graph G has a nested open ear decomposition starting from a path from s to t, then G is two terminal series parallel with terminals s and t.*

Proof: If G has only a single edge, then clearly it is series parallel. Otherwise there are two cases.

In the first case, there is some ear E_j, $j > 1$, having the same endpoints s and t as E_1. Then let Y be the graph induced by E_j and the ears contained in it, and let X be the graph induced by the remaining edges. By lemma 5, X and Y are connected only at s and t. The ears in Y form a nested ear decomposition starting with the path E_j, and the remaining ears in X form a nested ear decomposition starting from path E_1. By induction, X and Y are series parallel, and $G = P(X, Y)$.

In the second case, no such ear exists. Let E_j be an ear properly nested in E_1, such that the nest interval of E_j is not contained in any other nest interval. Then E_j has an endpoint x which is neither s nor t, since otherwise E_j would satisfy the conditions of the first case. Let X be the subpath of E_1 from s to x, together with all ears E_i contained in this subpath. Similarly let Y be the subpath of E_1 from x to t, together with all ears E_i contained in this subpath. By the nesting property and the maximality of E_j no ear nested in E_1 crosses E_j; thus all ears properly nested in E_1 are part of either X or Y. By induction we see that each ear E_i is part of one of the subgraphs, and no ear connects the two subgraphs, for each ear E_i must be nested in some previous ear, which in turn cannot connect the subgraphs. Further, X and Y have nested open ear decompositions starting from their respective subpaths of E_1. Thus $G = S(X, Y)$. $\square$

Lemma 7 *If $ED = \{E_1, E_2, \ldots E_k\}$ is an open ear decomposition of a two terminal series parallel graph G with terminals s and t, and with E_1 a path from s to t, then ED is nested.*

Proof: The proof is by induction on the number of series parallel composition operations making up G. If G consists of the single edge (s, t), then ED must consist of a single ear $E_1 = \{(s, t)\}$, which is clearly nested.

The next case is that $G = P(X, Y)$ for some X and Y. X and Y are connected only through s and t, so E_1 must be contained entirely within one or the other component. Further, each successive ear must again be contained in one or the other component, or else s or t would be an interior vertex of the ear, contradicting the properties of an ear decomposition. Assume without loss of generality that E_1 is contained in X. The ears in X form an open ear decomposition, and so by induction on the size of G they are nested. Now let E_j be the first ear with an edge in Y. Then the endpoints of E_j must be s and t, because those are the only vertices of Y that appear in previous ears. Thus the ears in Y again form an open ear decomposition starting from E_j, and again they are nested. No ear E_i from X can cross an ear E'_i in Y, because if they are both nested in the same ear, then their endpoints must be s and t and so their nest intervals must both be equal to the entire ear in which they are nested. Thus the entire ear decomposition is nested.

Finally, assume $G = S(X, Y)$, with X and Y meeting at vertex x. Then x disconnects G into two components, so E_1 must go through x and no other ear can contain edges in both X and Y. The path in E_1 from s to x together with the remaining ears in X forms an ear decomposition of X, which is therefore nested, and the path in E_1 from x to t together with the remaining ears in Y again forms a nested ear decomposition. No ear in X can cross one in Y, because if they are nested in any ear it must be ear E_1, and their nest intervals are contained in the disjoint subpaths from s to x and from x to t. Thus again the entire ear decomposition is nested. $\square$

Let us summarize the results of this section:

Theorem 1 *Any undirected two terminal series parallel graph has a nested ear decomposition starting with a path between the terminals, and any undirected graph with a nested ear decomposition is two terminal series parallel with its terminals being the endpoints of the first ear.* $\square$

3 Terminal Selection

Before we can use theorem 1 in an algorithm for recognizing series parallel graphs, we must first relate them to lemma 1 so that we can be sure of finding an open ear decomposition. Also, since our input graph will not have its terminals specified, we must show how to select a pair of vertices s and t such that, if G is series parallel, it is series parallel with those terminals.

Recall the well known fact that the set of *biconnected components* of a graph (maximal subgraphs that remain connected after the deletion of any vertex) forms a tree, having as its nodes the biconnected components and separating vertices of the graph, and with two nodes connected by an edge in the following two cases:

1. One node is a separating vertex, and the other is a biconnected component containing that vertex.

2. Both nodes are separating vertices connected by a single edge in the graph.

We now show some standard properties of series parallel graphs (e.g. see [6]).

Lemma 8 *If G is two terminal series parallel, with terminals s and t, then the tree of biconnected components of G must be a path, such that s and t are contained only in the components at the ends of the path (i.e. they can not be separating vertices).*

Proof: We show inductively that all the following facts hold:

1. The tree of biconnected components of G must be a path, with s and t properly contained in the components at the ends of the path.
2. If G is formed by a parallel composition operation, it is biconnected.
3. If v is a vertex of G, there exist vertex disjoint paths in G from s to v, and from v to t.
4. If v and w are vertices of G in different biconnected components then there exist vertex disjoint paths P_1, P_2, and P_3 connecting v to w and the two vertices with the two terminals, either v to s and w to t, or v to t and w to s.

The lemma clearly holds for a single edge.

Let $G = S(X, Y)$ meeting at point x. Then the biconnected components of G are those of X and Y, connected by a pair of edges corresponding to the cutpoint at x. Thus if the trees of components of X and Y are each paths, then the tree of G is also a path, and fact (1) holds. Fact (2) holds vacuously. To prove fact (3), without loss of generality assume that v is in X. Then there are vertex-disjoint paths from v to s, and v to x. Compose the latter path with any path from x to t to form a path from v to t that is disjoint from that from v to s. Finally we must prove fact (4). If both v and w are in the same subgraph, without loss of generality X, the paths of fact (4) exist simply by using the induction hypothesis of fact (4), and extending the path to vertex x by composing it with with any path from x to t in Y. Otherwise, assume without loss of generality that v is in X and w is in Y. By the induction hypothesis of fact (3), there are disjoint paths in X from s to v, and from v to x. Similarly there are disjoint paths in Y from x to

w, and from w to t. Now simply compose the paths from v through x to w, forming together with the remaining two paths the three disjoint paths required by fact (4). So the lemma holds for series composition.

Finally let $G = P(X, Y)$. Fact (1) is implied by fact (2), because the tree of biconnected components of a biconnected graph is trivially a path. To show that G is biconnected we need to find two vertex-disjoint paths between any two vertices v and w. If v and w are respectively in X and Y, use the induction hypothesis of fact (3) to find disjoint paths from v to s and v to t in X, and from w to s and w to t in Y. Compose the path from v to s with that from w to s in Y to form one path from v to w; and similarly compose the paths through t to form another path; these two paths must be vertex-disjoint. Otherwise, assume both v and w are in X, and use the induction hypothesis of fact (4) to find three disjoint paths, all in X, without loss of generality from s to v, v to w, and w to t. Then compose the paths from s to v and w to t with any path in Y from s to t, to produce a second disjoint path from v to w. Thus again the two vertices are connected by disjoint paths, and so the graph must be biconnected. To prove fact (3), without loss of generality for v in X, simply use the induction hypothesis of fact (3) for X. Fact (4) follows trivially from the biconnectedness of the graph. Thus all four facts hold for parallel as well as series composition, and so for all series parallel graphs. $\Box$

Lemma 9 *If G is a biconnected series parallel graph, and (s, t) is any edge in G, then G is two terminal series parallel with s and t as terminals.*

Proof: If G is a single edge, the lemma clearly holds. Otherwise, let G be series parallel with terminals v and w. Since G is biconnected it must be $P(X, Y)$ for some X and Y. Assume without loss of generality that (s, t) is in X. We prove the theorem by induction on the size of X. If X is a single edge, then again we are done. If $X = S(A, B)$, assume without loss of generality that (s, t) is in A. Then $G = P(A, S(B, Y))$, and A is smaller than X, so by induction the lemma holds. If $X = P(A, B)$, with (s, t) in A, then $G = P(A, P(B, Y))$ and again the lemma holds. $\Box$

Lemma 10 *If series parallel graph G is not biconnected, Let X and Y be the biconnected components at the endpoints of the path of biconnected components, let x and y be the cutpoints between their respective components and the remainder of the graph, and let s and t be any vertices in X and*

Y adjacent to x and y respectively. Then G is two terminal series parallel with terminals s and t.

Proof: By lemma 8, G must be the series composition of each of its biconnected components. By lemma 9, the components can be decomposed with terminals (s, x) and (y, t). Using these decompositions together with the serial composition of the remaining components gives back G, decomposed to have terminals s and t. $\square$

To summarize:

Theorem 2 *Any series parallel graph G is two terminal series parallel with terminals selected as in lemma 9 if G is biconnected, or as in lemma 10 otherwise.* $\square$

4 Detecting Nesting

We describe here the solution to a key subproblem in the recognition of series parallel graphs. In our recognition algorithm, we will have constructed an ear decomposition, which if the graph is series parallel must be nested. However we will need to verify this property in order to check whether the graph is in fact series parallel. If we wish to find a decomposition into the series and parallel composition operations, we further need to find the tree corresponding to the nesting structure. In fact we wish to create a nesting tree that also contains nodes for each of the edges in the outer ear, treated as if they were ears themselves. Therefore we now show how this can be done.

We can treat the collection of ears properly nested in a given ear independently of those ears nested in other ears. Form the *ear path graph H_i* for each ear E_i as follows: H_i contains a path composed of copies of the vertices in ear E_i; further, for each ear E_j properly nested in E_i we add an edge in H_i between the vertices that are copies of the endpoints of E_j. Note that H_i may have multiple edges with the same pairs of endpoints; this will not be a problem for our algorithms.

From now on in this section the words "edge" and "vertex" refer to edges and vertices in H_i, and not in the original input graph. A *path edge* is an edge of H_i corresponding to one of the edges in ear E_i. A *non-path edge* is any other edge of H_i; that is, an edge corresponding to an ear properly nested in E_i.

A subtle point in the construction of H_i is how to split the adjacency lists of vertices in the original graph into adjacency lists in the various graphs H_i. This can be solved by using the property that each vertex must be an interior vertex of at most one ear. We first split the adjacency lists into two sets: those edges corresponding to ears nested in the ear in which the vertex is interior, and those nested in other ears. Thus, for each graph H_i, we can compute the adjacency lists of all vertices except the copies of the endpoints of ear E_i. But these two remaining adjacency lists can be constructed in a prefix computation [1, 5, 10] simply by scanning all ears properly nested in E_i.

With these definitions, our problem becomes that of testing, for each H_i, whether the non-path edges of H_i nest, and if so constructing a nesting tree. We must take logarithmic time and a number of operations proportional to the size of H_i.

Number the vertices in H_i in order from one end of the path to the other. This can be done with a parallel list ranking algorithm [1, 5]. For each edge e in H_i, let MIN(e) be the smaller of the two numbers of its endpoints, and let MAX(e) be the larger of the two numbers.

For each vertex, split its list of adjacent edges into two lists: the edges going backwards (to a vertex with a lower number) and the edges going forwards (to a vertex with a higher number). We assume that the path edges are listed first in each case. Concatenate all the lists of backwards edges, from lower numbered endpoints to higher numbered ones. This gives a list of all edges in H_i, sorted in order by their values of MAX(e). For a given value of MAX(e), the path edge having that value comes before the other edges with the same value.

For each edge e in this list, let NEST(e) be the nearest edge f appearing before e in the list with MIN(f) $\leq$ MIN(e); this can be computed with the "All Nearest Smaller Values" algorithm of Berkman et al. [2]. We now show that NEST(e) captures the nesting order of the edges of H_i.

First we must define a unique nesting order. The only ambiguous case is when for two edges e and f of H_i, MIN(e) = MIN(f) and MAX(e) = MAX(f). Then if f appears earlier in the list of edges than e, we say that f is nested in e. This unambiguously and consistently resolves any possible ambiguity. By this criterion, all path edges are nested within any edges sharing the same pairs of endpoints, and all non-path edges have at least a path edge nested within them; this is the reason we listed path edges before other edges with the same values of MAX(e).

We say that an edge g is nested *directly* within edge e if g is nested in e, but there is no edge f with g nested in f and f nested in e.

Lemma 11 *Let the graph H_i be nested. Then for each non-path edge e,* $\mathrm{MIN}(e) = \mathrm{MIN}(\mathrm{NEST}(e))$*, and* $\mathrm{NEST}(e)$ *is the unique edge nested directly within e and sharing the same value of* MIN*.*

Proof: First assume that, $\mathrm{MIN}(\mathrm{NEST}(e)) < \mathrm{MIN}(e)$. Then let f be the path edge with $\mathrm{MIN}(f) = \mathrm{MIN}(e)$. If $\mathrm{MIN}(\mathrm{NEST}(e)) < \mathrm{MIN}(e)$, then by the assumption that H_i is nested, $\mathrm{MAX}(\mathrm{NEST}(e)) \leq \mathrm{MIN}(e) < \mathrm{MAX}(f)$, and so $\mathrm{NEST}(e)$ occurs before f in the list of edges. But, since f was not chosen as $\mathrm{NEST}(e)$, it must not occur before e in the list, and the only way this can happen is that $e = f$.

So if e is a non-path edge, $\mathrm{MIN}(e) = \mathrm{MIN}(\mathrm{NEST}(e))$. It follows that $\mathrm{NEST}(e)$ is nested in e.

The nearest edge in the list having the same value of MIN will be one with the largest value of MAX. If two such edges exist, the later one in the list will be chosen. Therefore, by our disambiguating rule, it follows that $\mathrm{NEST}(e)$ is directly nested in e. $\square$

Theorem 3 *Given an ear decomposition of a graph, we can test whether the decomposition is nested, and if so construct the tree corresponding to the nesting structure within each ear, in time $O(\log n)$ with $O(n)$ parallel operations.*

Proof: In fact we compute the nesting trees and then use them to verify that the decomposition is nested. Therefore we now construct a nesting tree, with nodes corresponding to the edges of H_i, and links corresponding to the nesting of those edges. We will later verify that our construction actually gives us a nesting tree.

We first create a node for each edge of H_i, with one extra root node. Our tree will be specified by listing, for each node, its first child and its next sibling. All other information can be recovered from these two links.

Lemma 11 lets us determine the first child of each parent node. The non-parent nodes are exactly those corresponding to the path edges. It remains to find the next sibling of each node.

First note that the node corresponding to edge e has a next sibling if and only if e is not nested within another ear f with $\mathrm{MAX}(e) = \mathrm{MAX}(f)$. Symmetrically, e has a previous sibling if and only if it is not nested within another ear f with $\mathrm{MIN}(e) = \mathrm{MIN}(f)$.

Therefore for each vertex v interior to path H_i, there is exactly one pair of edges e and f such that $\mathrm{MAX}(e) = \mathrm{MIN}(f) = v$, and such that f is the next sibling of e. The edge f can be found by searching the adjacency list of v for the edge appearing last in the sorted list of edges. The edge e can be found symmetrically; for this we need to construct a different sorted list of all edges by concatenating the adjacency lists of the path vertices in reversed order.

The root of the nesting tree must be added as a new node, with first child the edge e with $\mathrm{MIN}(e)$ the first vertex in the path, and of all such edges the one appearing last in the sorted list.

All these steps can be performed for any decomposition, nested or not, yielding a first child and next sibling for some of the nodes in the tree. If H_i is nested, each edge in H_i must occur exactly once either as a first child or as a next sibling; this can be tested as follows. We allocate an array of memory cells corresponding to the nodes in the tree. Each node first clears its own cell. Then each node concurrently writes its name in the cell for its first child, and for its next sibling, if it has either of those two links. Third, each node verifies that no other node concurrently wrote to those two cells. Fourth, each node other than the root node verifies that some other node wrote a name in its cell.

The chains of nodes found by following the next sibling links cannot be cyclic. Therefore each first child of a parent can send its parent's identity along these chains, by a prefix computation. After this stage every node will know its parent, and it can easily be verified that each node is nested in its parent.

If all these conditions hold, we will have constructed a graph that is acyclic (because of the nesting property) and with a number of edges one fewer than the number of nodes (the specially constructed root node has no incoming edges). Therefore this graph must be a tree, and the decomposition must be nested. $\square$

5 The Algorithm

We now show how our results may be combined into a series parallel graph recognition algorithm. We should note a related concept of series parallel graphs, in which the terminals are not required to be disjoint; such graphs can be formed by including an isolated vertex in the class of series parallel graphs. It seems likely that our algorithms can be modified to recognize such graphs by using arbitrary ear decompositions, instead of open ear decompositions; however it seems easier to simply verify that each biconnected component of such a graph is series parallel in the usual sense.

Let $C(m,n)$ be the number of processors required to compute the connected components and spanning forest of a graph with m edges and n vertices, in time $O(\log n)$. The best bound known for this problem is $C(m,n) = O(m\alpha(m,n)/\log n)$ [5]. This differs from the best possible bound by at most the inverse Ackermann function. It is not known whether linear work can be achieved in logarithmic time.

To recognize an undirected series parallel graph, we perform the following steps on the given graph G. We will achieve our desired time bound if each step takes time $O(\log n)$, and a number of operations (time multiplied by processors) no more than $O(C(m,n)\log n)$. Several of the steps assume the graph is specified as lists of the edges incident to each vertex. If the graph is specified instead as one list of all the edges, a sorting step can be performed to transform it into the desired representation, but this will take $O(n)$ processors.

1. Find the tree of biconnected components of G using the algorithm of Tarjan and Vishkin [13]. This takes time $O(\log n)$ and uses $C(m,n)$ processors on a CRCW machine.

2. If G is biconnected, let (s,t) be any edge in G. Otherwise, verify that the tree of biconnected components is a path (i.e., G is connected and each component is adjacent to at most two others). Let v and w be the cutpoints of the end components in the path, and (s,v) and (t,w) be edges in those components. Add edge (s,t) to the graph, so that G is now biconnected. By the lemmas of the previous section, we will now have a pair of adjacent vertices s and t such that, if G is series parallel, s and t can be terminals of G. All these steps can be done in constant time with linearly many processors.

3. Find an open ear decomposition ED of G starting from the ear consisting of the single edge (s,t). This can be done by the algorithm of Maon et al [11], which again takes logarithmic time and $C(m,n)$ processors. If G is series parallel, by lemma 7, ED will be nested; now we must verify whether this is the case.

4. For each vertex v, compute $e_{ED}(v)$, the ear in which v is interior. For each ear E_i, compute $e_{ED}(E_i)$, the ear properly containing E_i, as the

greater of the two values of e_{ED} applied to the endpoints of E_i. These steps take constant time with linearly many processors.

5. Verify that the decomposition is nested, and form the nesting tree for each ear as in theorem 3. This step takes time $O(\log n)$ with $O((n+m)/\log n)$ processors.

6. Form the decomposition into series parallel composition operations as follows. Allocate place holders for the following symbols: $W(E_i)$ for each ear E_i; $X(e_j)$, $Y(e_j)$, and $Z(e_j)$ for each node e_j in the nesting tree for each ear E_i. If e_j corresponds to ear E_j, let $X(e_j) = W(E_j)$; otherwise e_j corresponds to an edge in ear E_i, and we let $X(e_j)$ be that edge. If e_j has first child e_k, let $Y(e_j) = P(X(e_j), Z(e_k))$; if e_j has no child let $Y(e_j) = X(e_j)$. If e_j has next sibling e_k let $Z(e_j) = S(Y(e_j), Z(e_k))$; otherwise let $Z(e_j) = Y(e_j)$. Finally for each ear E_i let $W(E_i) = Z(x)$ where x is the root of the nesting tree for E_i. All these steps can be performed in constant time with linearly many processors, and compressing the resulting decomposition to yield only the composition operations and not the equivalences between placeholders can be done in time $O(\log n)$ with $O((n+m)/\log n)$ processors.

To recognize an undirected series parallel graph with specified terminals, we simply omit the first two steps above. To recognize a directed series parallel graph, we take the terminals to be the unique source and sink of the graph, perform the steps above, and verify that each edge in the graph is directed in the appropriate direction.

To reconcile the differences between times and numbers of processors used by each step above, recall that any parallel algorithm can be slowed down to any time no slower than the total amount of work without changing the number of operations by more than a constant factor [4]. In our case the work of each step is at least linear and the slowdown is at most logarithmic, so there is no problem applying this result.

Putting the bounds for these steps together, we have

Theorem 4 *If G is a series parallel graph, specified by lists of edges incident to each vertex, that fact can be verified, and a series parallel decomposition of G constructed, in time $O(\log n)$ with $C(m,n)$ processors on a CRCW machine.* $\square$

Acknowledgements

This research was performed in part at Columbia University, supported in part by NSF grants DCR-85-11713 and CCR-86-05353, and by DARPA contract N00039-84-C-0165.

I would like to thank Baruch Schieber, for persuading me to publish these results, and an anonymous referee, for suggestions leading to the improvement of the processor bounds for my algorithms.

References

[1] R.J. Anderson and G.L. Miller, Deterministic Parallel List Ranking, 3rd Aegean Workshop on Computing, Springer-Verlag LNCS 319 (1988) 91–100.

[2] O. Berkman, D. Breslauer, Z. Galil, B. Schieber, and U. Vishkin, Highly Parallelizable Problems, Proc. 21st ACM Symp. Theory of Computing (1989) 309–319.

[3] M.W. Bern, E.L. Lawler, and A.L. Wong, Linear-time Computation of Optimal Subgraphs of Decomposable Graphs, J. Algorithms 8(2) (1987) 216–235; a preliminary version appeared as Why Certain Subgraph Computations Require Only Linear Time, Proc. 26th IEEE Symp. Foundations of Computer Science (1985) 117–125.

[4] R.P. Brent, The Parallel Evaluation of General Arithmetic Expressions, J. ACM 21 (1974) 201–206.

[5] R. Cole and U. Vishkin, Approximate and Exact Parallel Scheduling with Applications to Optimal Parallel List Ranking, Info. and Control 70 (1986) 32–53.

[6] R.J. Duffin, Topology of Series Parallel Networks, J. Math. and Appl. 10 (1965) 303–318.

[7] X. He and Y. Yesha, Parallel Recognition and Decomposition of Two Terminal Series Parallel Graphs, Info. and Comput. 75 (1987) 15–38.

[8] Samir Khuller, Ear Decompositions, SIGACT News 20(1) (1989) 128.

[9] Tohru Kikuno, Noriyoshi Yoshida, and Yoshiaki Kakuda, A Linear Algorithm for the Domination Number of a Series Parallel Graph, Disc. Appl. Math. 5 (1983) 299–311.

[10] R.E. Ladner and M.J. Fischer, Parallel Prefix Computation, J. ACM 27 (1980) 831–838.

[11] Y. Maon, B. Schieber, and U. Vishkin, Parallel Ear Decomposition Search (EDS) and ST-Numbering in Graphs, Theor. Comput. Sci. 47 (1986) 277–198, and VLSI Algorithms and Architectures, Springer-Verlag LNCS 227 (1986) 34–45.

[12] K. Takamizawa, T. Nishizeki, and N. Saito, Linear-Time Computability of Combinatorial Problems on Series-Parallel Graphs, J. ACM 29 (1982) 623–641.

[13] R.E. Tarjan and U. Vishkin, An Efficient Parallel Biconnectivity Algorithm, SIAM J. Comput. 14(4) (1985) 862–874; a preliminary version appeared as Finding Biconnected Components and Computing Tree Functions in Logarithmic Parallel Time, Proc. 25th IEEE Symp. Foundations of Computer Science (1984) 12–20.

[14] Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler, The Recognition of Series Parallel Digraphs, SIAM J. Comput. 11 (1982), 289–313, and Proc. 11th ACM Symp. Theory of Computing (1979) 1–12.

[15] H. Whitney, Non-Separable and Planar Graphs, Trans. Amer. Math. Soc. 34 (1932) 339–362.