Parallel algorithms for series parallel graphs[*]

Hans L. Bodlaender and Babette de Fluiter
Department of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands
e-mail: {hansb,babette}@cs.ruu.nl

Abstract

In this paper, a parallel algorithm is given that, given a graph $G = (V, E)$, decides whether G is a series parallel graph, and if so, builds a decomposition tree for G of series and parallel composition rules. The algorithm uses $O(\log |E| \log^* |E|)$ time and $O(|E|)$ operations on an EREW PRAM, and $O(\log |E|)$ time and $O(|E|)$ operations on a CRCW PRAM (note that if G is a simple series parallel graph, then $|E| = O(|V|)$). With the same time and processor resources, a tree-decomposition of width at most two can be built of a given series parallel graph, and hence, very efficient parallel algorithms can be found for a large number of graph problems on series parallel graphs, including many well known problems, e.g., all problems that can be stated in monadic second order logic. The results hold for undirected series parallel graphs graphs, as well as for directed series parallel graphs.

1 Introduction

One of the classical classes of graphs is the class of *series parallel graphs.* These appear in several applications, e.g., the classical way to compute the resistance of an (electrical) network of resistors assumes that the underlying graph is in fact a series parallel graph.

A well studied problem is the problem to recognise series parallel graphs. A linear time algorithm for this problem has been given by Valdes, Tarjan, and Lawler [11]. Also, it is known that when a 'decomposition tree' for a series parallel graph is given, then many problems can be solved in linear time, including many problems that are NP-hard for arbitrary graphs [2, 5, 9, 10]; Valdes et al. also show how to obtain such a decomposition tree in linear time. (In this paper, we assume a specific form of the decomposition tree, and use the term *sp-trees* for these trees.)

He and Yesha gave a parallel algorithm for recognising directed series parallel graphs [8]. Their algorithm uses $O(\log^2 n + \log m)$ time, and $O(n + m)$ processors on an EREW PRAM, thus, with $O((n + m)(\log^2 n + \log m))$ operations. (The number of operations of a parallel algorithm is the product of its time and number of processors used. We measure parallel algorithms with the number of operations, as this gives us a direct comparison with the best sequential algorithm. In this paper, n denotes the number of vertices of the input graph; m the

[*]This research was partially supported by the Foundation for Computer Science (S.I.O.N) of the Netherlands Organisation for Scientific Research (N.W.O.).

number of edges.) Eppstein [7] improved this result for simple graphs: his algorithm runs in $O(\log n)$ time on a CRCW PRAM with $O(m\alpha(m,n)/\log n)$ processors. As any algorithm on a CRCW PRAM can be simulated on an EREW with a loss of $O(\log n)$ time, this implies an algorithm with $O(\log^2 n)$ time and $O(m\alpha(m,n)/\log n)$ processors on an EREW PRAM, i.e., with $O(m\alpha(m,n))$ operations on an EREW PRAM, or with $O(m\alpha(m,n)\log n)$ operations on a CRCW PRAM.

In this paper, we improve upon this result, both for the EREW PRAM model, and for the CRCW PRAM model. We give a recognition algorithm for series parallel graphs on the EREW PRAM, that uses $O(\log m \log^* m)$ time and $O(m)$ operations, and one for the CRCW PRAM, that uses $O(\log m)$ time and $O(m)$ operations. In the same time and operations bounds, we can also build an sp-tree for the graph, and solve many problems on series parallel graphs. If the input graph is simple, i.e. there is at most one edge between every two vertices, then $m = O(n)$ if the graph is series parallel. If we are sure that the input graph is a simple graph, then we can make our algorithm to run in $O(\log n \log^* n)$ on the EREW PRAM and $O(\log n)$ on the CRCW PRAM, and the number of operations is $O(n)$.

It is well-known that series parallel graphs have treewidth at most two. (Some authors mistakenly state that the class of series parallel graphs equals the class of graphs of treewidth two, but for instance $K_{1,3}$ is not series parallel.) We will use this fact in one of our proofs. Moreover, several of our results were inspired by techniques, established for graphs of bounded treewidth, especially those from [3] and [4]. As a side remark, we note that, while the algorithms in [4] are carrying constant factors that make them impractical in their stated form, the algorithms in this paper do not carry large constant factors and are probably efficient enough for a practical setting (although a more detailed analysis can probably bring the constant factor further down.)

A central technique in this paper is *graph reduction*, introduced in a setting of graphs of bounded treewidth in [1]. In [3] and [4], it is shown how the technique can be used to obtain parallel algorithms for graphs of bounded treewidth.

Another technique that is used in this paper is the *bounded adjacency list search* technique, taken from [4], and adapted here to the setting of series parallel graphs.

This paper is organised further as follows. In Section 2, we give some basic definitions, and preliminary results. In Section 3, we give a number of graph reduction rules, and show that they are 'safe' for series parallel graphs. In Section 4, we show that in any series parallel graph, there are $\Omega(m)$ reduction rules that can be applied concurrently. Section 5 shows the main algorithm, based on the results of the earlier sections. Section 6 gives some results for variants of the problem, and shows how to solve many other problems on series parallel graphs.

2 Definitions and preliminary results

Unless stated otherwise, graphs considered are undirected, and may have parallel edges.

Definition 2.1. *A* series parallel graph *is a triple (G, s, t), with $G = (V, E)$ a graph, and $s, t \in V$, such that one of the following cases holds.*

- *G has two vertices, s and t, and one edge between s and t.*

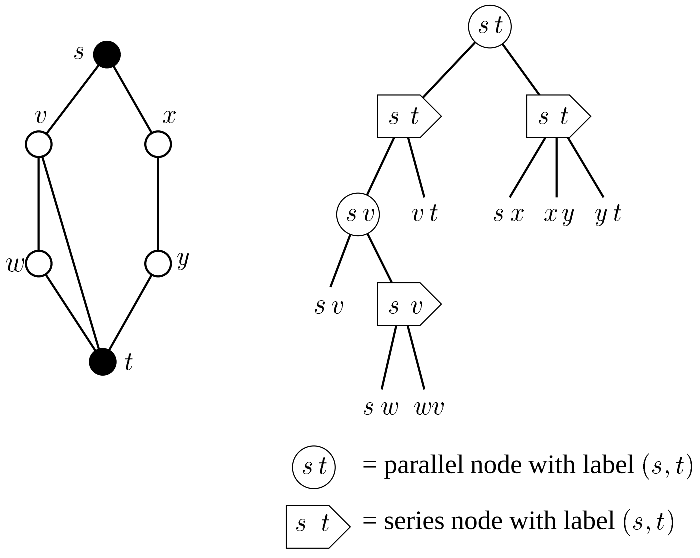

s
v
x
w
y
t
s t
s t
s t
s v
v t
s x
x y
y t
s v
s v
s w
w v
s t = parallel node with label (s, t)
s t = series node with label (s, t)

Figure 1: An sp-tree and the series parallel graph it represents

- *There are series parallel graphs* (G_1, s_1, t_1), …, (G_r, s_r, t_r), *such that* (G, s, t) *can be obtained by a* parallel composition *of these graphs:* G *is obtained by taking the disjoint union of* $G_1, \ldots, G_r$, *identifying all vertices* $s_1, \ldots, s_r$ *to* s, *and identifying all vertices* $t_1, \ldots t_r$ *to* t.
- *There are series parallel graphs* (G_1, s_1, t_1), …, (G_r, s_r, t_r), *such that* (G, s, t) *can be obtained by a* series composition *of these graphs:* G *is obtained by taking the disjoint union of* $G_1, \ldots, G_r$, *identifying for* $i = 1, \ldots, r \Leftrightarrow 1$, *vertex* t_i *with vertex* s_{i+1}, *and letting* $s = s_1$, $t = t_r$.

If (G, s, t) is a series parallel graph, we also say that G is a series parallel graph. s and t are called the terminals of G; we also call s the *source* and t the *sink* of G.

See Figure 1 for an example. An equivalent definition, which is often used, only involves series and parallel compositions with two series parallel graphs.

To each series parallel graph $G = (V, E)$, we can associate an *sp-tree* T_G. Every node of an sp-tree corresponds to a series parallel graph (G', a, b), and has as a label (a, b), the ordered pair of its source and sink. The root of the tree has label (s, t), and corresponds to the graph (G, s, t). Leaf nodes correspond to series parallel graphs consisting of a single edge, and hence there is a one-to-one correspondence between leaves of T_G and edges $e \in E$. Internal nodes (i.e. non-leaf nodes) are of two types: series nodes (or s-nodes), and parallel nodes (or p-nodes). The children of a series node are ordered, while the children of a parallel node are not ordered. Note that the children of a p-node have the same label as their parent. The series parallel graph associated to an s-node is the graph, obtained by a series composition of the series parallel graphs, associated to the children of the node, where the order of the children gives the order in which the series composition is done. The series parallel graph associated to a p-node is the graph, obtained by a parallel composition of the series parallel graphs, associated to the children of G.

Note that a series parallel graph can have different sp-trees. However, if a source and sink are given, there also is a unique *minimal sp-tree*: given any sp-tree, if there is an s-node with another s-node as child, or a p-node with another p-node as child, one can contract over the parent-child edge, and obtain a smaller sp-tree for the same graph. Moreover, s-nodes or p-nodes with only one child can easily be removed. So, in the minimal sp-tree, s-nodes and p-nodes alternate.

Lemma 2.1. *If α is an ancestor of β in an sp-tree T_G, and the labels of α and β both contain a vertex v, then all nodes on the path between α and β in T_G contain v as label.*

We can also define directed series parallel graphs. These are defined in the same way as undirected series parallel graphs, with the sole exception that as 'base graph', we take a directed graph with two vertices s and t and a directed edge from the source s to the sink t. As a result, directed series parallel graphs are acyclic, and every vertex lies on a directed path from the source to the sink.

Definition 2.2. *A* tree-decomposition *of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\},\ T = (I, F))$, with $\{X_i \mid i \in I\}$ a collection of subsets of G, and T a tree, such that*

- $\bigcup_{i \in I} X_i = V$.
- *For all $\{v, w\} \in E$: there exists an $i \in I$ with $v, w \in X_i$.*
- *For all $v \in V$: the set $\{i \in I \mid v \in X_i\}$ induces a connected part (subtree) of T.*

The treewidth *of a tree-decomposition $(\{X_i \mid i \in I\},\ T = (I, F))$ is $\max_{i \in I} |X_i| \Leftrightarrow 1$. The treewidth of a graph G is the minimum treewidth over all possible tree-decompositions of G.*

To be able to describe the reduction rules of our algorithm, we introduce the notion of *k-terminal graph*.

A *terminal graph* is a triple (V, E, X), where (V, E) is a graph, and X is a subset of the vertices from V, numbered from 1 to $|X|$. X is called the set of terminals of (V, E, X). A vertex $v \in V$ is a *terminal* of (V, E, X) if $v \in X$, and is an *inner vertex* of (V, E, X), if $v \in V \Leftrightarrow X$. A terminal graph (V, E, X) is a k-terminal graph, if $|X| = k$.

The binary operation $\oplus$ is defined on terminal graphs with the same number of terminals. If $G_1 = (V_1, E_1, X_1)$ and $G_2 = (V_2, E_2, X_2)$ are k-terminal graphs, then $G_1 \oplus G_2$ is the graph, obtained by taking the disjoint union of (V_1, E_1) and (V_2, E_2), and then identifying the jth terminal in X_1 and X_2, for all j, $1 \leq j \leq k$.

Two k-terminal graphs $(V_1, E_1, (x_1, \ldots, x_k))$ and $(V_2, E_2, (y_1, \ldots, y_k))$ are said to be *isomorphic*, if there exist bijective functions $f : V_1 \rightarrow V_2$, $g : E_1 \rightarrow E_2$, such that for all $v \in V_1$, $e \in E_1$: v is an endpoint of e, if and only if $f(v)$ is an endpoint of $g(e)$, and for all i, $1 \leq i \leq k$, $f(x_i) = y_i$).

While series parallel graphs are a special type of two-terminal graphs, to avoid confusion, we will see series parallel graphs as graphs with two vertices with a special label.

A *source-sink* labelled graph is a graph $G = (V, E)$, with two distinct vertices labelled, one with the label *source*, and one with the label *sink*. A source-sink labelled graph G, with s

labelled source, and t labelled sink is said to be series parallel, if (G, s, t) is a series parallel graph.

We now give some simple or well known lemmas.

Lemma 2.2. *If $(G = (V, E), s, t)$ is a series parallel graph, then $(G + \{s, t\}, s, t)$ is a series parallel graph, where $G + \{s, t\}$ is the graph, obtained by adding an edge $\{s, t\}$ to G.*

Proof. By doing a parallel composition of G with a one-edge series parallel graph. $\square$

Lemma 2.3. *If $(G = (V, E), s, t)$ is a series parallel graph with sp-tree T_G, and there is a node α in T_G, labelled with (w, x), then $(G + \{w, x\}, s, t)$ is a series parallel graph.*

Proof. Suppose G_α is the series parallel graph, associated with node α. Add between α and its parent a p-node, with two children: α and a leaf node, representing edge $\{w, x\}$. $\square$

Lemma 2.4. *Let T be an sp-tree of series parallel graph $G = (V, E)$; let $x, y \in V$. The set of nodes $\{i \mid i$ is labelled with $(x, y)\}$ induces a (possibly empty) subtree of T.*

The following lemma is given with a constructive proof, as this will be useful later for obtaining some algorithmic results.

Lemma 2.5. *If G is a series parallel graph, then the treewidth of G is at most two.*

Proof. First note, that any sp-tree can be transformed (using standard transformation techniques) to a binary sp-tree (of the same graph). Suppose $T_G = (N, F)$ is a binary sp-tree. For a parallel node i with label (v, w), write $X_i = \{v, w\}$; for a series node with label (v, w) and labels of its two children (v, x) and (x, w), write $X_i = \{v, w, x\}$. Now, one can verify that $(\{X_i \mid i \in N\}, T_G)$ is a tree-decomposition of G of treewidth at most two. $\square$

A graph $G = (V, E)$ is said to be a *minor* of a graph $H = (W, F)$, if a graph, isomorphic to G can be obtained from H by a series of vertex deletions, edge deletions, and edge contractions.

Lemma 2.6. *If the treewidth of G is at most two, then G does not contain K_4 as a minor.*

Lemma 2.7. *Let $G = (V, E)$ be a series parallel graph with source s and sink t.*

1. *If there is a node α with label (x, y) in an sp-tree of G, then there is a simple path $s, \ldots, x, \ldots, y, \ldots, t$ in G.*
2. *If there is a node with label (x, y) in an sp-tree of G that is an ancestor of a node with label (v, w), then there is a simple path $s, \ldots, x, \ldots, v, \ldots, w, \ldots, y, \ldots, t$ in G.*
3. *For every edge $e = \{x, y\} \in E$, there is either a simple path $s, \ldots, x, y, \ldots, t$, or there is a simple path $s, \ldots, y, x, \ldots, t$.*

Proof. 1. We prove that for any node β with label (v, w) on the path from α to the root of the sp-tree of G, there is a simple path $v, \ldots, x, \ldots, y, \ldots, w$, that uses only edges whose corresponding nodes are descendants of β. We use induction to the length of the path from α to β in the sp-tree. (Using this result with β the root of the sp-tree gives the desired result.)

First, suppose $\alpha = \beta$. As any series parallel graph is connected, there is a path from v to w in the series parallel graph associated with node α.

Next, suppose β is an ancestor of α. Let γ be the child of β on the path from α to β. If β is a p-node, then the label of β equals the label of γ and the result directly follows. Suppose β is an s-node with children $\delta_1, \ldots, \delta_r$, and δ_i has label (v_i, v_{i+1}) for each i, $1 \leq i \leq r$. Let j, $1 \leq j \leq r$, be such that $\delta_j = \gamma$.

Now, for any i, $1 \leq i \leq r$, there is a simple path from v_i to v_{i+1} using only edges that are a descendant of δ_i. By induction hypothesis, there is a simple path from v_j to v_{j+1} of the form $v_j, \ldots, x, \ldots, y, \ldots, v_{j+1}$ that uses only edges that are descendants of δ_j. Concatening these paths gives the required simple path of the form $v_1 = v, \ldots, x, \ldots, y, \ldots, w = v_{r+1}$.

2. Similar.

3. Note that there is a node with label (x, y) or a node with label (y, x). Now we can use part 1 of this Lemma. □

Lemma 2.8. *Let there be a simple path $s, \ldots, x, y, \ldots, t$ in a series parallel graph $G = (V, E)$ with source s and sink t.*

1. *Either there is no simple path from s to y that avoids x, or there is no simple path from x to t that avoids y.*
2. *No node in an sp-tree of G is labelled with the pair (y, x).*

Proof. 1. Suppose not. Then $(G + \{s, t\}, s, t)$ contains K_4 as a minor, which contradicts Lemma 2.6.

2. This follows from part 1 of this lemma, and Lemma 2.7. □

Lemma 2.9. *Suppose $\{x, y\}$ is an edge in a series parallel graph $G = (V, E)$ with source s and sink t. Suppose there is a simple path in G $s, \ldots, x, y, \ldots, t$. Let W be the set*

$$W = \{v \in V \Leftrightarrow \{x, y\} \mid \textit{there is a simple path } s, \ldots, x, \ldots, v, \ldots, y, \ldots, t \textit{ in } G\}.$$

Then:

1. *For all $\{v, w\} \in E$, $v \in W$ implies that $w \in W \cup \{x, y\}$.*
2. *For every sp-tree of G, if a node is labelled with endpoints v, w, and $v \in W$, then $w \in W \cup \{x, y\}$.*
3. *Let T be an sp-tree of G, let α be the highest node with label (x, y). The series parallel graph G_α associated with α is exactly the graph $G[W \cup \{x, y\}]$. Furthermore, if $|W| \geq 1$, then α is a parallel node.*

Proof. 1. Suppose $\{v, w\} \in E$, $v \in W$, $w \notin \{x, y\}$. By Lemma 2.7, one of the following two cases must hold.

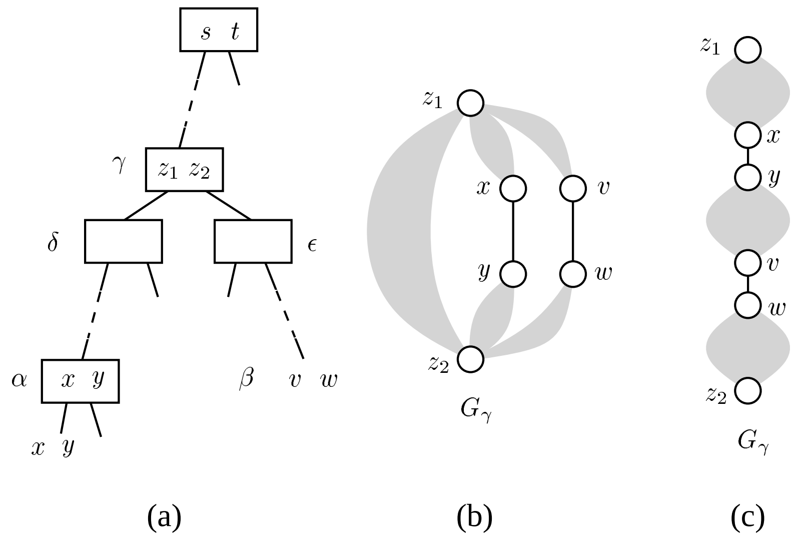


Figure 2: The sp-tree and possible graphs for the proof of Lemma 2.9

Case 1. There is a simple path $s, \ldots, v, w, \ldots, t$. If x and y do both not appear on the part of the path from s to v, then $G + \{s, t\}$ contains K_4 as a minor, contradiction. Hence either x or y belongs to the path from s to v. Similarly, x or y belongs to the part of the path from w to t. If y appears on the first part, and x appears on the last part, then we have a contradiction with Lemma 2.8. Hence, we have a simple path of the form $s, \ldots, x, \ldots, v, w, \ldots, y, \ldots, t$. This implies that $w \in W$.

Case 2. There is a simple path $s, \ldots, w, v, \ldots, t$. This case is similar to Case 1.

2. Note that if a node in the sp-tree of G is labelled with (v, w), then $G + \{v, w\}$ is also a series parallel graph (Lemma 2.3). Hence, the result follows by part 1 of the lemma.

3. We first show that G_α is a subgraph of $G[W \cup \{x, y\}]$. Let $v \in V(G_\alpha)$. There is a descendant β of α which contains v in its label. According to Lemma 2.7, there is a simple path $s, \ldots, x, \ldots, v, \ldots, y, \ldots, t$, so $v \in W$.

Next we show that $G[W \cup \{x, y\}]$ is a subgraph of G_α. Let $e = \{v, w\} \in E(G[W \cup \{x, y\}])$, let β be the leaf node of e, and suppose w.l.o.g. that β has label (v, w). We show that β is a descendant of α. If $e = \{x, y\}$, this clearly holds.

Suppose $e \neq \{x, y\}$ and β is not a descendant of α. Then we have a node γ, with label $(z_1, z_2) \neq (x, y)$, with children δ and ϵ, such that α is equal to or a descendant of δ, and β is equal to or a descendant of ϵ (see Figure 2(a)).

If $z_1 \in W$, then G contains a path from s to x that avoids z_1, and G contains a path from z_1 to y that avoids x. Also, G contains a simple path $s, \ldots, z_1, \ldots, x, y$, hence $G + \{s, t\}$ contains a K_4 minor, contradiction. So, we may assume that $z_1 \notin W$, and similarly, that $z_2 \notin W$.

First suppose that γ is a p-node. Figure 2(b) shows the structure of the series parallel graph G_γ associated with node γ. The graph G_ϵ associated with ϵ contains a simple path

$z_1, \ldots, x, y, \ldots, z_2$, because of Lemma 2.7, part 2. Similarly, the graph G_δ associated with node δ contains a simple path $z_1, \ldots, v, w, \ldots, z_2$. Since the only common vertices of G_ϵ and G_δ are z_1 and z_2, there is a simple path $x, \ldots, z_1, \ldots, v, \ldots, z_2, \ldots, y$ in G. Since $(x, y) \neq (z_1, z_2)$ and $z_1, z_2 \notin W$, this means that this path contains an edge between a vertex in W and a vertex in $V \Leftrightarrow W \Leftrightarrow \{x, y\}$, which is in contradiction with part 1 of this lemma.

Suppose γ is an s-node, and suppose that node δ is on the left side of node ϵ. Figure 2(c) shows the structure of the series parallel graph G_γ. There is no path $z, \ldots, v, \ldots, y$ in G_γ, which means that any path in G which goes from x to y and contains v must look like $x, \ldots, z_1, \ldots, z_2, \ldots, v, \ldots, y$. This again means that there is an edge between a vertex in W and a vertex in $V \Leftrightarrow W \Leftrightarrow \{x, y\}$, contradiction. If δ is on the right side of ϵ, then in the same way, we have a path $x, \ldots, v, \ldots, z_1, \ldots, z_2, \ldots, y$. This is again a contradiction. Hence β is a descendant of α. This proves that $G_\alpha = G[W \cup \{x, w\}]$.

If α is an s-node, then it is the only node with label (x, y). This is impossible, because there is a leaf node with label (x, y). If α is a leaf node, then G_α consists only of the edge $\{x, y\}$. Hence if $|W| \geq 1$, then α is a p-node. This completes the proof of part 3. □

3 Graph reductions

In this section, we consider a number of graph reduction operations.

A *reduction rule* is an ordered pair (H_1, H_2), where H_1 and H_2 are k-terminal graphs for some $k \geq 0$. An *application* of reduction rule (H_1, H_2) is the operation, that takes a graph G of the form $G_1 \oplus G_3$, with G_1 isomorphic to H_1, and replaces it by the graph $G_2 \oplus G_3$, with G_2 isomorphic to H_2. We call such a rule application a *reduction*.

A reduction rule is *safe* for a property P, if, whenever G is obtained from H by applying the rule, $P(G) \Leftrightarrow P(H)$. A set of rules is *safe* for P, if all rules in the set are safe.

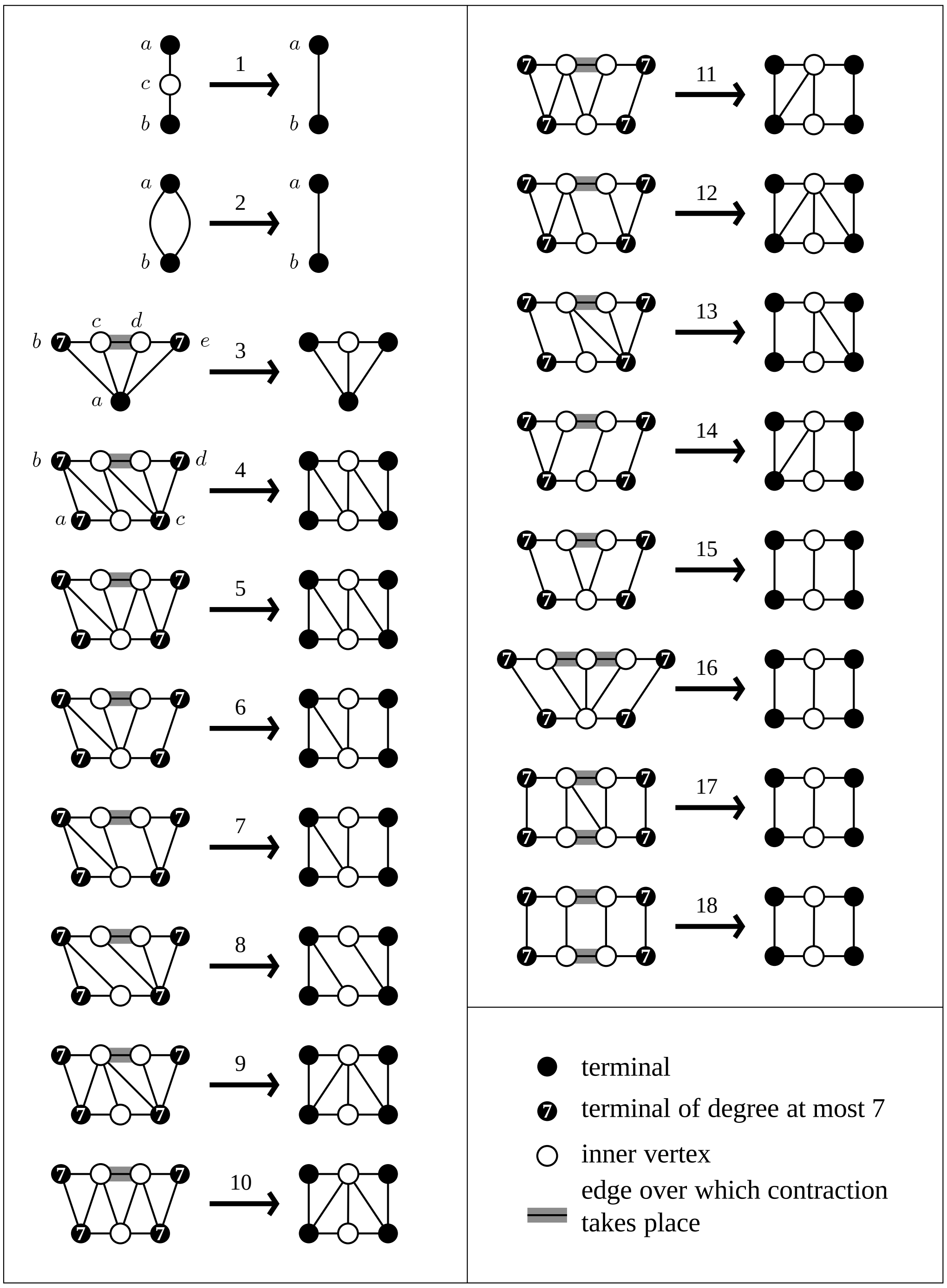
The notion of reductions is generalised in the natural manner to source-sink labelled graphs. In this case, it is assumed that no inner vertex of a left-hand side or right-hand side graph of a rule is a vertex with a source or sink label.

We now give a set of 18 reduction rules, that is safe for the class of series parallel graphs. Additionally, we pose some degree restrictions on terminal vertices in some of the rules. In the next section, we show that we can always find a 'sufficiently large' set of reduction rules of the given types, such that all rules in this set can be applied concurrently, until the graph consists of a single edge.

Note that each of the rules described below, is performed on a source-sink labelled graph. We depict the rules in Figure 3. Inner vertices are depicted by unfilled circles, terminal vertices are depicted by filled circles; a number inside a terminal vertex denotes an upper bound on the degree of the vertex.

Rule 1. Remove a non-terminal vertex of degree two, and add an edge between its neighbours.

Lemma 3.1. *If (G', s, t) is obtained from (G, s, t) by applying Rule 1, then (G, s, t) is a series parallel graph if and only if (G', s, t) is a series parallel graph.*



a
c
b
1
a
b
a
b
2
a
b
b
c
d
e
a
3
b
d
a
c
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
● terminal
7 terminal of degree at most 7
○ inner vertex
edge over which contraction takes place

Figure 3: Reduction rules for series parallel graphs

Proof. Suppose G' is obtained by removing vertex c of degree two, suppose the neighbours of c are a and b. Suppose we have an sp-tree for G'. There must be a leaf with label (a,b) or label (b,a). Suppose w.l.o.g. that there is a leaf node with label (a,b). If the parent of this leaf is an s-node, then replace this leaf node by two leaf nodes, successively (a,c) and (c,b). If the parent of this leaf is an p-node, then the leaf is replaced by a s-node with two children, (a,c) and (c,b).

Now, suppose we have a minimal sp-tree for G. As c is not a terminal, there must be a series composition that composed $\{b,c\}$ and $\{c,a\}$. So, the modification above can be reversed. □

Rule 2. Remove one of two parallel edges.

Lemma 3.2. *If (G',s,t) is obtained from (G,s,t) by applying Rule 2, then (G,s,t) is a series parallel graph if and only if (G',s,t) is a series parallel graph.*

Proof. Suppose (G',s,t) is obtained by removing edge e_2 from G, where e_2 is parallel to edge e_1. If we have an sp-tree for G', then this tree has a leaf node α which corresponds to e_2 (and hence the end points of e_1 are in its label). If the parent of α is a p-node, then attach an additional leave below this parent, with the same label as α. If the parent of e_1 is an s-node, then replace α by a p-node, with two children: one for e_1 and one for e_2.

When we have an sp-tree for G, remove the leaf node corresponding to e_2 from this tree. When now the (former) parent, say β, of this leaf node has only one child, this child is directly attached to the parent of β. □

Rule 3. Rule 3 is depicted in Figure 3. Note that edges between terminals can have parallel edges, but edges with at least one endpoint an inner vertex in a left-hand side of a rule cannot have a parallel edge.

Lemma 3.3. *Suppose (G',s,t) is obtained from (G,s,t) by one application of Rule 3. Then, (G,s,t) is a series parallel graph if and only if (G',s,t) is a series parallel graph.*

Proof. Suppose (G,s,t) is a series parallel graph, and let T be the minimal sp-tree of (G,s,t). Consider a simple path P from s to t that uses the edge $\{a,b\}$.

Case 1. We first suppose that the path P visits a before b. We distinguish between two further cases.

Case 1.1. First, we suppose that P does not use the node e. Write

$$\begin{aligned} W &= \{v \in V \mid \text{there is a simple path } s,\ldots,a,\ldots,v,\ldots,b,\ldots,t \text{ and} \\ &\qquad v \text{ belongs to the same component as } e \text{ in } G[V \Leftrightarrow \{a,b\}]\}. \end{aligned}$$

Note that $c,d,e \in W$, and hence (by part 1 of Lemma 2.9), all vertices in the component of $G[V \Leftrightarrow \{a,b\}]$ which contains e are in W. There must be a parallel node α in T with label (a,b), with the subgraph containing the nodes in W 'below it' (see part 3 of Lemma 2.9). Each vertex $v \neq a, b$ of the graph G_α associated with α can occur in at most one graph associated with one of the children of α.

Let β be the s-node that is a child of α such that the series parallel graph G_β associated with β contains e. We claim that G_β is the graph obtained from $G[W \cup \{a, b\}]$ by deleting all edges between a and b. If a vertex $w \in W$ is not in G_β, then all paths from w to e use a or b, which means that w is not in the component of $G[V \Leftrightarrow \{a, b\}]$ which contains e. Hence $w \in V(G_\beta$. Hence each vertex of W occurs only in G_β, which means that all edges between vertices in W and in $W \cup \{a, b\}$ are in G_β.

On the other hand, if there is a vertex $x \in G_\beta$, $x \notin \{a, b\}$, then there is a path $P = a, \ldots, x, \ldots, b$ in G_β (Lemma 2.7). If P contains no vertex from W, then β can not be a series node. Hence P contains a vertex from W. Together with part 1 of Lemma 2.9, this means that all vertices on P are in $W \cup \{a, b\}$, so $x \in W$. The graph G_β can not contain an edge between a and b, since then β can not be an s-node. This proves the claim.

Suppose β has children with labels (a, x_1),$(x_1, x_2), \ldots, (x_t, b)$, respectively. We show that $t = 1$ and $x_1 = x_t = c$. Suppose not. First suppose that $x_t \neq c$. Add an edge between x_t and b; this again gives a series parallel graph. Now, by contracting all nodes in W except c to d, we get a K_4 minor, contradiction. Hence $x_t = c$. Now suppose that $t > 1$. There is a leaf node with label (a, c) or label (c, a) which is a descendant of β, since there is an edge $\{a, c\}$. But vertex a occurs only in the labels of the subtree of the child of β with label (a, x_1). Furthermore, vertex c occurs only in the labels of the subtrees of the children of β with labels (c, b) and (x_{t-1}, c). Since $x_1 \neq c$ and $x_{t-1} \neq a$, this means that there can be no leaf node with label (a, c) or (c, a), which gives a contradiction. So $t = 1$, the children of β have labels (a, c) and (c, b), respectively. It can be seen that the child with label (c, b) is a leaf node, corresponding to edge $\{b, c\}$. By straightforward deduction, it follows that the sp-tree of G has the tree from the left-hand side of Figure 4(i) as a subtree. We can replace this subtree by the subtree shown in the right-hand side of Figure 4(i) and get an sp-tree of G'.

Case 1.2. Now, we suppose the simple path P from s to t that uses the edge $\{a, b\}$, also uses node e. There are a two different cases:

Case 1.2.1. Path P is of the form $s, \ldots, e, (\ldots,) a, b, \ldots, t$. Now, $G + \{s, t\}$ is series parallel, but contains K_4 as a minor, contradiction.

Case 1.2.2. Path P is of the form $s, \ldots, a, b, \ldots, e, \ldots, t$. Now, we have a simple path $s, \ldots, a, e, \ldots, t$, that does not use b. This case can be analysed in exactly the same way as the cases above, leading to a subtree transformation, as shown in Figure 4(iii).

Case 2. We now suppose that the path P visits b before a. This case can be dealt with in the same way as Case 1, only with directions reversed. See Figures 4(ii),(iv).

This ends 'only if' part of the proof. The 'if' part is very similar. In this case, the same transformations as above are done, but in opposite direction. □

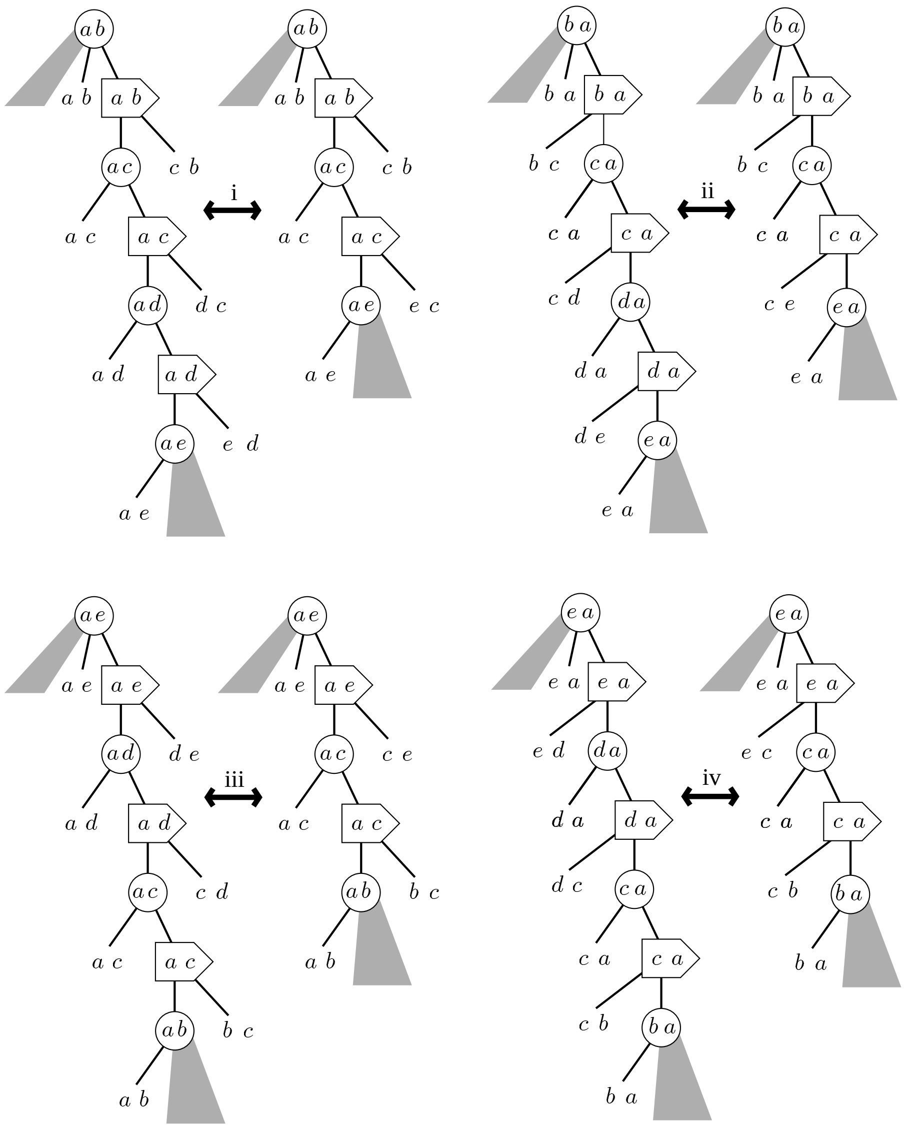


Figure 4: Transformations of subtrees for Rule 3.

Rules 4 – 18 Rules 4 – 18 are depicted in Figure 3.

Lemma 3.4. *Suppose (G', s, t) is obtained from (G, s, t) by one application of of one of the Rules 4 – 18. Then, (G, s, t) is a series parallel graph, if and only if (G', s, t) is a series parallel graph.*

Proof. The proof is similar to the proof of Lemma 3.3. As an example, consider Rule 4. Suppose (G, s, t) is a series parallel graph, and let T be a minimal sp-tree of (G, s, t). Consider a simple path P from s to t that uses the edge $\{a, b\}$. First suppose P visits a before b. We distinguish four cases.

Case 1. P does not use vertices c and d. We can show that (G', s, t) is series parallel in the same way as in Case 1.1 in the proof of Lemma 3.3 (define W to be the component of $G[V \Leftrightarrow \{a, b\}]$ which contains c and d).

Case 2. P uses c but not d. Then either c is on the sub-path $s, \ldots, a$ of P or c is on the sub-path $b, \ldots, t$ of P. In both cases, $G + \{s, t\}$ contains a K_4 minor, which gives a contradiction.

Case 3. P uses d but not c. This case is similar to Case 2, and hence gives a contradiction.

Case 4. P uses both c and d. If c and d both occur on the sub-path $s, \ldots, a$ of P, or on the sub-path $b, \ldots, t$ of P, then $G + \{s, t\}$ contains a K_4 minor.

If $P = s, \ldots, d, \ldots, a, b, \ldots, c, \ldots, t$, then $G + \{s, t\}$ also contains a K_4 minor.

If $P = s, \ldots, c, \ldots, a, b, \ldots, d, \ldots, t$ then there is a simple path from s to t that uses the edge $\{c, d\}$, and does not use a and b. This case is similar to Case 1.

The case that P visits vertex b before a can be solved in the same way. This ends the 'only if' part of the proof. The 'if' part can be handled in the same way.

For the proof of Rules 5–18, we can apply exactly the same technique. □

An important consequence of the proofs of Lemmas 3.1 – 3.4 is that the proofs are constructive: especially, when we have a minimal sp-tree of the reduced graph, we can build, in $O(1)$ time, a minimal sp-tree of the original graph.

4 A structural lemma

A set of reductions is said to be *concurrent*, if no inner vertex of any subgraph to be rewritten also occurs in another subgraph to be rewritten. So the subgraphs to be rewritten can share terminals. Note that it is possible to carry out all reductions of a concurrent set of reductions simultaneously.

Lemma 4.1. *Let $G = (V, E)$ be a series parallel graph. There is a concurrent set of at least $|E|/204$ reductions in G of Rules 1 — 18.*

Proof. Consider the minimal sp-tree T of G. The number of leaves of T equals $|E|$. We argue that the number of leaves of T is at most equal to 204 times the number of concurrent applications of reduction rules. Therefore, we consider different 'classes' of leaves.

A leaf node α in T is *good* if it is a child of a parallel node and has at least one sibling which is a leaf (i.e. α is child of a parallel node which has at least two leaf children), or it is a child of a series node and one of α's neighbouring siblings also is a leaf node (i.e. α is child of a series node which has at least two successive leaf children of which α is one). Note that the edges that correspond to good leaf nodes can occur in the application of reduction Rule 1 or 2.

An internal node in T is *green* if it has at least one good leaf child.

A node in T is *branching* if it is an internal node, and has at least two internal nodes as its children.

A leaf is *bad* if it is not good, and its parent is branching or green. Most edges that correspond to bad leaves can not occur in any application of a reduction rule.

Note that the leaf children of a branching node which is not green are all bad, and the leaf children of a green node are either bad or good.

Now consider the other leaf nodes in T. An internal node is *blue* if it is not branching or green, but it has a descendant that is branching or green at distance at most 33.

An internal node is *yellow* if it is not branching, green or blue.

The total number of leaves in T equals the number of good leaves plus the number of bad leaves plus the number of leaf children of blue nodes plus the number of leaf children of yellow nodes. We now derive an upper bound for the number of leaves in each of these classes, in terms of the number of concurrent applications of reduction rules.

Good leaves. If a green p-node has m good leaves, then we can concurrently apply at least $\lfloor m/2 \rfloor$ times reduction Rule 2 on the edges corresponding to these leaves. If a green s-node has m good leaves, then we can concurrently apply at least $\lfloor m/3 \rfloor$ times reduction Rule 1 on the edges corresponding to these leaves. Hence the number of good leaves is at most three times the number of concurrent applications of reduction Rules 1 and 2.

Bad leaves.

Claim 4.1. The number of bad leaves is at most twice the number of branching nodes plus the number of green nodes plus the number of blue nodes.

Proof. Let α be a bad leaf. If α's parent is a parallel node, then account α to its parent (which has at most one bad leaf). If α's parent is a series node, then account α to its neighbouring sibling on the right if it has one, or to its parent otherwise. In this way, each branching node has at most two leaves accounted to it: at most one of its children and possibly its neighbouring sibling on the left. Each green, yellow or blue node has at most one bad leaf accounted to it, namely its neighbouring sibling on the left. If a yellow node β has a bad leaf accounted to it, then its highest blue descendant γ has no bad leaf accounted to it, since it does not have a branching parent. Furthermore, all yellow nodes on the path from β to γ have no bad leaf accounted to it, since they have no branching parents. Hence we can account the bad leaf that is accounted to α, to β. This means that the number of bad leaves accounted to yellow and blue nodes is at most equal to the number of blue nodes. This proves the claim. □

In each green node, we can apply Rule 1 or 2 on two of the edges corresponding to its good leaves. Hence the number of green nodes is at most equal to the number of concurrent applications of Rules 1 and 2. We now bound the number of branching and blue nodes by the number of green nodes in order to bound the number of bad leaves.

Claim 4.2. The number of branching nodes is at most equal to the number of green nodes.

Proof. Make tree T' from T, by removing all nodes that are not green and not branching, while preserving successor-relationships. Note that, in T, every internal node that has only leaves as child is green, hence every branching node still has at least two children in T'. Moreover, every leaf of T' is green. Since the number of internal nodes in a tree with two or more children is at most the number of leaves, the number of branching nodes is at most the number of green nodes in T', and hence in T. □

Consider the number of blue nodes. The number of blue nodes is at most 33 times the number of branching and green nodes: account each blue node to the closest descendant which is branching or green. Since the number of branching nodes is at most the number of green nodes, this means that the number of blue nodes is at most $2 \cdot 33 = 66$ times the number of green nodes.

This means that the number of bad leaves is at most equal to $2 + 1 + 66 = 69$ times the number of green nodes, which is at most 69 times the number of concurrent applications of Rules 1 and 2.

Leaves of blue nodes. The number of blue nodes is at most 66 times the number of green nodes. Each blue node has at most two leaf children, which means that the number of leaves of blue nodes is at most $2 \cdot 66 = 132$ times the number of concurrent applications of Rules 1 and 2.

Leaves of yellow nodes. Consider a path in T which consists of 33 successive yellow and blue nodes, such that the highest node in this path is a parallel node. Each node in this path either is a p-node with as its children one leaf node and one s-node, or it is an s-node with as its children one p-node and one or two non-neighbouring leaf nodes.

The edges associated to the leaves that are a child of the nodes in this path form a subgraph of G of a special form: they form a sequence of 16 cycles of length three or four, each sharing one edge with the previous cycle, and one edge with the next (except of course for the first and last cycle in the sequence); three successive cycles do not share one common edge. As no series node on the path has two successive leaf nodes, we have that the shared edges of a cycle of length four do not have a vertex in common. We call such a subgraph a *cycle-sequence*. See Figure 5 for an example.

Claim 4.3. In a cycle-sequence which consists of 16 cycles, one of the Rules 3 – 18 can be applied.

Proof. Consider a sequence of 33 successive yellow and blue nodes starting and ending with a p-node, and its corresponding cycle-sequence. Let $\bar{a} = a_1, a_2, \ldots, a_s$, $\bar{b} = b_1, b_2, \ldots, b_t$, such that each node on the path of yellow and blue nodes has label (a_i, b_j), $1 \leq i \leq s$, $1 \leq j \leq t$,

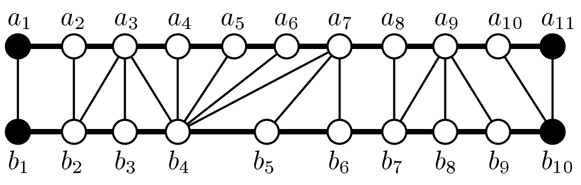


Figure 5: Subgraph of G corresponding to a path of 33 yellow or blue nodes in the sp-tree, of which the highest one is a p-node with label (a_1, b_1), and the lowest one is a p-node with label (a_{11}, b_{10}). Only a_1, b_1, a_{11} and b_{10} may be incident with edges outside the subgraph.

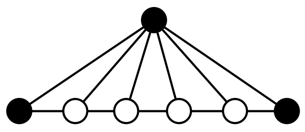


Figure 6: Five successive triangles with one vertex in common

and for each $1 \leq i < i' \leq s$ ($1 \leq j < j' \leq t$), $a_i \neq a_{i'}$ ($b_j \neq b_{j'}$), and the highest node containing $a_{i'}$ ($b_{j'}$) in its label is a descendant of the lowest node containing a_i (b_j) in its label (hence the highest node has label (a_1, b_1), the lowest node has label (a_s, b_t), and all a_i and b_j occur at least once in a label). Let $P_1 = a_1, a_2, \ldots, a_s$ and let $P_2 = b_1, b_2, \ldots, b_t$. Note that P_1 and P_2 are paths in G. We call them the *bounding paths* of the cycle-sequence (see Figure 5).

If the cycle-sequence contains a cycle-sequence of five successive triangles with one vertex in common as its subsequence, as in Figure 6, then Rule 3 can be applied. Suppose such a subsequence does not exist. It follows that the edge between the fifth and sixth three- or four-cycle in the sequence does not have an endpoint that is also endpoint of an edge not in the subgraph; similarly for the edge between the 11th and 12th cycle. Moreover, all vertices that belong to the sixth till 11th cycle belong to at most six three- or four-cycles, and hence have degree at most seven. Consider the cycle-sequence formed by the sixth till 11th cycle. Let P_1' and P_2' be the bounding paths of this cycle-sequence, suppose P_1' has length (i.e. number of edges) m and P_2' has length n. We now show that we can apply one of the reduction Rules 4 – 18 on this cycle-sequence.

Suppose w.l.o.g. that $m \leq n$. We first show that $m \geq 2$ and $n \geq 3$. If $m = 0$, then the only vertex of P_1' occurs in six triangles. If $m = 1$ then one of the vertices of P_1' occurs in three triangles. Hence if $m \leq 1$ then we can apply Rule 3 to the cycle-sequence. If $m = n = 2$, then the cycle-sequence consists of at most four cycles. Hence $n \geq 3$ and $m \geq 2$.

The left-hand sides of Rules 4 – 15 represent exactly the cycle-sequences with one bounding path of length two and one of length three which can not be reduced by applying Rule 3. The left-hand sides of Rules 17 and 18 represent exactly the cycle-sequences with two bounding paths of length three which can not be reduced by applying one of the Rules 3 – 15. Hence if our cycle-sequence contains a subsequence with one bounding path of length three and one of length two or three, then we can apply one of the Rules 3 – 15, 17 or 18.

Now suppose the cycle-sequence does not contain such a subsequence. We show that

Rule 16 is applicable. The shortest of the two bounding paths has length at least two and the longest one has length at least three. Remove one of the outermost cycles of the cycle-sequence until one of these conditions would be violated by removing another outermost cycle. Let P_1'' be the shortest bounding path and P_2'' the longest bounding path of the obtained cycle-sequence.

If P_1'' has length three, then P_2'' must have length three, otherwise we can remove another outer-cycle. But that means that one of the Rules 4 – 15, 17 and 18 can be applied. Hence P_1'' has length two. If P_2'' has length three, then one of the Rules 4 – 15 can be applied, hence P_2'' has length four or more. If its length is five or more, then Rule 3 is applicable, hence its length is four. Then the outermost cycles must be squares, otherwise Rule 3 is again applicable. But that means that the cycle-sequence is equal to the left-hand side of Rule 16. This proves the claim. □

In a sequence of 34 successive yellow and blue nodes in T, we can find one path of 33 successive yellow and blue nodes, such that the highest node in this path is a p-node. We can find a number of disjoint paths of 34 successive yellow and blue nodes, such that each yellow node is in exactly one such path. This means that the largest number of disjoint paths of successive yellow and blue nodes of length 34 that we can find in T is at least $1/34$ times the number of yellow nodes. Hence the number of concurrent applications of Rules 3 – 18 is at least $1/34$ times the number of yellow nodes. This means that the number of leaf children of yellow nodes is at least $2 \cdot 34 = 68$ times the number of concurrent applications of Rules 3 – 18.

The total number of leaves in T is now at most $3 + 69 + 132 = 204$ times the number of concurrent applications of Rules 1 and 2 plus 68 times the number of concurrent applications of Rules 3 – 18. Since the applications of Rules 1 and 2 do not interfere with the applications of the other Rules, we have that the number of leaves in T is at most 204 times the number of concurrent applications of reduction Rules in G. This completes the proof. □

5 Main algorithm

In this section, we give the main algorithm. Suppose we have given an undirected, not necessarily simple, graph G with two specified vertices s and t, and we want to determine if (G, s, t) is series parallel, and if so, we want to build an sp-tree for G. The algorithm consists of two main phases. The first phase consists of $O(\log m)$ reduction rounds. In each reduction round, a number of reductions is carried out, each round (when the input is a series parallel graph) reducing the number of edges of G with at least a constant fraction. In the first phase, the input graph is reduced to a single edge $\{s, t\}$ if and only if it is series parallel. If (G, s, t) is not series parallel, i.e., we do not have a single edge after the first phase, then the algorithms stops. Otherwise, we proceed with the second phase. In the second phase, all reductions are undone, in an equally large number of rounds. During the ‘undoing’ of the reductions, we maintain a minimal sp-tree of the current graph. (One can additionally also maintain a binary sp-tree of the current graph.)

A round in the first phase consists of a few steps. First, every edge ‘looks around’ to see whether it can take part in a reduction. Here, possibly not all possible reductions are found, but at least ‘a large enough number of possible reductions’ are found. Then, a subset of the reductions is selected, such that these reductions do not create conflicts when carried out simultaneously. Finally, the reductions are done — some bookkeeping is done such that later the reductions can be undone.

Which edges can take part in an application of Rules 1, or 3 – 18, can easily be determined, by only looking at adjacency lists of nodes of degree at most seven, in $O(1)$ time per node. In a reduction step, all possible choices for reductions of these rules are found. However, we will not find all possible choices for Rule 2 reductions: this is probably not possible in the given time bounds in the EREW PRAM model. Instead, every edge looks in both adjacency lists of its endpoints to those edges that have distance at most ten, and the edge proposes a reduction, if one of these edges it looks at has the same endpoints. Thus, this rule application can also be carried out in $O(1)$ time per edge. (Adjacency lists are assumed to be cyclic.)

Each reduction found in this way is said to be *enabled*. We now show that $\Omega(|E|)$ reductions are enabled.

Lemma 5.1. *If $G = (V, E)$ is a simple series parallel graph, then $|E| \leq 2|V|$.*

Say an edge is *bad*, if it has a parallel edge, but no parallel edge is found in the procedure above.

Lemma 5.2. *If (G, s, t) is series parallel, then there are at most $2|E|/5$ bad edges.*

Proof. Consider a tree-decomposition $(\{X_i \mid i \in I\}, T)$ of G of width two, and choose an arbitrary node $i \in I$ as root of T. For a $v \in V$, let r_v be the highest node in T with $v \in X_{r_v}$. If $\{v, w\} \in E$, then note that either $r_v = r_w$, or r_v is an ancestor of r_w, or r_w is an ancestor of r_v, as there is a node with labels v and w.

For every bad edge $\{v, w\}$, associate the edge with v if $r_v = r_w$, or r_w is an ancestor of r_v; otherwise, associate the edge with w. Suppose bad edge $\{v, w\}$ is associated with v. Now, X_{r_v} must contain both v and w. It follows that there are at most $|X_{r_v}| \Leftrightarrow 1 \leq 2$ different vertices w, such that bad edges $\{v, w\}$ can be associated with v (namely, the vertices in $X_{r_v} \Leftrightarrow \{v\}$). For each such w, each ten successive positions in the (cyclic) adjacency list of v can contain at most one bad edge of the form $\{v, w\}$, hence there are at most $\mathrm{degree}(v)/10$ bad edges $\{v, w\}$ associated with v, and hence in total, at most $\mathrm{degree}(v)/5$ bad edges are associated with v. The stated bound is derived by taking the sum over all vertices. □

Now, if $|E| \geq 4n$, then there are at least $|E| \Leftrightarrow 2n$ edges that are parallel to another edge, of which at most $2|E|/5$ are bad. Hence, $3/5|E| \Leftrightarrow 2n \geq 3/5|E| \Leftrightarrow 1/2|E| = |E|/10$ edges find a parallel edge. Hence $\Omega(E)$ reductions are enabled.

Now, suppose $|E| \leq 4n$. We now apply Lemma 4.1 above on the simple graph underlying G. I.e., let G' be obtained from G by removing all second and further occurrences of parallel edges. Note that G' has at least $n \Leftrightarrow 1$ edges. (We ignore the simple case that G' consists of a single edge in the remainder.) Hence, there are at least $(n \Leftrightarrow 1)/204$ reductions possible on G', and as G' has no parallel edges, each of these is of Rules 1 or 3 – 18. For each reduction in this set, there are two possibilities: either the reduction is also possible in G, or there are parallel edges between two vertices that are both involved in the reduction. But, as at least one of these two vertices has degree at most seven in G', at least some parallel edges will be detected with this node as endpoint. This shows that at least $(n \Leftrightarrow 1)/204$ possible reductions will be enabled, but as $|E| \leq 4n$, these are $\Omega(|E|)$ reductions.

As subgraphs that are involved in enabled rule applications may overlap, it is not possible to carry out all enabled rule applications simultaneously. Also, some reduction-pairs would try to simultaneously write or read to a memory location. Thus, it is needed to find a large set of reductions that can be carried out simultaneously, without any conflicts arising. This is solved in the same way as the reduction algorithms in [4] are done: a 'conflict graph' is built; one can note that this conflict graph has bounded degree, and a large independent set in the conflict graph is then found. By using the same approach as in [4], we can carry out all reductions in $O(\log m \cdot \log^* m)$ time with $O(m)$ operations and $O(m)$ space on an EREW PRAM, and with $O(\log m)$ time and $O(m)$ operations and $O(m)$ space on a CRCW PRAM.

This situation is handled further in exactly the same way as in [4].

As each reduction round reduces the number of edges with a constant fraction when the input is a series parallel graph, after $O(\log m)$ reduction rounds we can conclude whether the input was series parallel or not, depending on whether we end up with a single edge or not. (Note that all reductions are safe, i.e., we start with a series parallel graph if and only if we end with a series parallel graph.)

The second phase builds the sp-tree, in case (G, s, t) was series parallel. The sp-tree is represented as follows. Each node in the tree has a mark denoting its type (series, parallel or leaf), a mark containing its label, a pointer to its parent, and a pointer to a doubly linked list of its children (in the correct order if it is a series node). Furthermore, each vertex v in the graph has a pointer to one of the nodes in the sp-tree that contains v in its label.

We start with the simple sp-tree, with a single node, labelled (s, t). Then, we undo each reduction round. Given an sp-tree for the reduced graph, we build the sp-tree for the graph as it was just before this reduction round was carried out in the first phase, using the constructions as in the proofs of Lemmas 3.1 – 3.4. The processor that carried out the reduction in the first round will be the same processor that carries out the undoing of the reduction. Note that each undoing of a single reduction can be done in $O(1)$ time without concurrent reading or writing.

A small modification to the construction also allows us not only to maintain a minimal sp-tree, but also a binary sp-tree.

This technique was based on work, reported in [3], where also more details can be found.

Theorem 5.1. *The following problem can be solved in $O(m)$ operations, and $O(\log m \log^* m)$ time on a EREW PRAM, and $O(\log m)$ time on a CRCW PRAM: given a graph $G = (V, E)$, and $s, t \in V$, determine whether G is series parallel with source s and sink t, and if so, find a minimal or binary sp-tree.*

6 Additional results for series parallel graphs

The algorithm, given in the previous section can also be used to solve the recognition problem for directed series parallel graphs, and for series parallel graphs without specified source and sink. Also, it can be used as a first step to solve many other problems on series parallel graphs.

First, suppose we are given a graph $G = (V, E)$, and want to determine whether G is series parallel with a proper choice of terminals. In [7], it was shown (using results from [6]) that this problem reduces in a direct way to the problem with specified vertices, as the following result holds.

Theorem 6.1. [Eppstein [7]] *Let $G = (V, E)$ be an undirected graph. If there exist vertices $r, q \in V$, such that (G, r, q) is series parallel, then (G, s, t) is series parallel, with source $s \in V$ and sink $t \in V$ chosen in the following way.*

- *If G is biconnected, then s and t are chosen to be adjacent vertices.*
- *If G is not biconnected, then there must be exactly two biconnected components that contain only one cut vertex. Source s is taken to be a vertex in one such biconnected component, such that s is adjacent to the cut vertex of the biconnected component. Sink t is taken in the same way in the other biconnected component with one cut vertex.*

Now, note that the characterisation of s and t as in Theorem 6.1 above can be formulated in monadic second order logic (using techniques from e.g., [5]); hence, it is possible (using techniques of [4, 3]) to find values of s and t which fulfil the conditions of Theorem 6.1 in $O(\log m \log^* m)$ time, with $O(m)$ operations and space on an EREW PRAM, and in $O(\log m)$ time, and $O(m)$ operations and space on a CRCW PRAM. While the resulting algorithm will probably not be efficient, this result does not rely on non-constructive arguing. (We expect that a more straightforward approach, based on reduction, will also work here.)

When G is directed, then one can use the modification, described in [7]: solve the problem first on the underlying undirected graph, and then verify that all edges have the proper direction. If s and t are not specified, then take for s the vertex with indegree 0, and for t the vertex with outdegree 0.

Theorem 6.2. *Each of the following problems can be solved in $O(m)$ operations, and $O(\log m \log^* m)$ time on an EREW PRAM, and $O(\log m)$ time on a CRCW PRAM.*

1. *Given a graph $G = (V, E)$, determine if there exist $s \in V$, $t \in V$, such that G is series parallel with source s and sink t, and if so, find a corresponding sp-tree.*
2. *Given a directed graph $G = (V, E)$, and $s, t \in V$, determine whether G is series parallel with source s and sink t, and if so, find a corresponding sp-tree.*
3. *Given a directed graph $G = (V, E)$, determine if there exist $s \in V$, $t \in V$, such that G is series parallel with source s and sink t, and if so, find a corresponding sp-tree.*

Using the connection with treewidth, it is possible to design a large class of algorithms, solving problems on series parallel graphs. Many problems can be solved in $O(\log n)$ time, and $O(n)$ operations and space, when the input graph is simple, and is given together with a tree-decomposition of bounded treewidth. These include all problems that can be formulated in monadic second order logic and its extensions, all problems that are 'finite state', etc. A large number of interesting and important graph problems can be dealt in this way, including CHROMATIC NUMBER, MAXIMUM CLIQUE, MAXIMUM INDEPENDENT SET, HAMILTONIAN CIRCUIT, STEINER TREE, LONGEST PATH, etc. See [4].

Now, note that we can build a tree-decomposition of treewidth two of a given series parallel graph in the following way: first make a binary sp-tree, and then use the construction of Lemma 2.5.

Lemma 6.1. *One can find a tree-decomposition of treewidth two of a series parallel graph in $O(\log m \log^* m)$ time, $O(m)$ operations, and $O(m)$ space on an EREW PRAM, and $O(\log m)$ time, $O(m)$ operations and $O(m)$ space on a CRCW PRAM.*

As a consequence, a very large class of graph problems can be solved in the same time bounds.

Acknowledgement

We like to thank Torben Hagerup for help and useful discussions.

References

[1] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *J. ACM*, 40:1134–1164, 1993.

[2] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear time computation of optimal subgraphs of decomposable graphs. *J. Algorithms*, 8:216–235, 1987.

[3] H. L. Bodlaender and B. de Fluiter. Reduction algorithms for graphs with small treewidth. Technical Report UU-CS-1995-37, Department of Computer Science, Utrecht University, Utrecht, 1995.

[4] H. L. Bodlaender and T. Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. In Z. Fülöp and F. Gécseg, editors, *Proceedings 22nd International Colloquium on Automata, Languages and Programming*, pages 268–279, Berlin, 1995. Springer-Verlag, Lecture Notes in Computer Science 944.

[5] R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992.

[6] R. J. Duffin. Topology of series-parallel graphs. *J. Math. Anal. Appl.*, 10:303–318, 1965.

[7] D. Eppstein. Parallel recognition of series parallel graphs. *Information and Computation*, 98:41–55, 1992.

[8] X. He and Y. Yesha. Parallel recognition and decomposition of two terminal series parallel graphs. *Information and Computation*, 75:15–38, 1987.

[9] T. Kikuno, N. Yoshida, and Y. Kakuda. A linear algorithm for the domination number of a series-parallel graph. *Disc. Appl. Math.*, 5:299–311, 1983.

[10] K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *J. ACM*, 29:623–641, 1982.

[11] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. *SIAM J. Comput.*, 11:298–313, 1982.