Parallel Algorithms for Series Parallel Graphs*

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

Babette de Fluiter
Centre for Quantitative Methods
P.O. Box 414, 5600 AK Eindhoven, the Netherlands
e-mail: deFluiter@cqm.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. 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. These include many well-known problems like all problems that can be stated in monadic second order logic. The results hold for undirected series parallel graphs as well as for directed series parallel graphs.

1 Introduction

One of the well known classes of graphs is the class of *series parallel graphs*. Series parallel graphs 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 series parallel.

A well-studied problem is the problem to recognize series parallel graphs. A linear time algorithm for this problem has been given by Valdes, Tarjan, and Lawler [15]. 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 [3, 6, 13, 14]; 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-tree* for this type of decomposition tree.)

*This research was carried out while the second author was working at the Department of Computer Science of Utrecht University, with support by the Foundation for Computer Science (S.I.O.N) of the Netherlands Organization for Scientific Research (N.W.O.). This research was partially supported by ESPRIT Long Term Research Project 20244 (project ALCOM IT: *Algorithms and Complexity in Information Technology*).

He and Yesha [12] and He [11] gave parallel algorithms for recognizing directed and undirected series parallel graphs in $O(\log^2 n + \log m)$ time with $O(n+m)$ processors on an EREW PRAM, and hence $O((n+m)(\log^2 n + \log m))$ operations. (The number of operations of a parallel algorithm is the product of its time and number of processors used. In this paper, n denotes the number of vertices of the input graph; m the number of edges.) Their algorithm also returns a decomposition tree of the input graph, if it is series parallel.

Eppstein [10] improved these results for simple graphs: his algorithms run in $O(\log n)$ time on a CRCW PRAM with $O(m \cdot \alpha(m,n))$ operations ($\alpha(m,n)$ is the inverse of Ackermann's function, which is at most four for all practical purposes). As any algorithm on a CRCW PRAM can be simulated on an EREW PRAM with a loss of $O(\log n)$ time, this implies an algorithm with $O(\log^2 n)$ time and $O(m \log n \cdot \alpha(m,n))$ operations on an EREW PRAM.

We improve upon these results, both for the EREW PRAM model and the CRCW PRAM model: we give algorithms for recognizing directed and undirected series parallel graphs, and building a decomposition tree if one exists. These algorithms use $O(\log m \log^* m)$ time with $O(m)$ operations on an EREW PRAM, and in $O(\log m)$ time with $O(m)$ operations on a CRCW PRAM. If the input graph is simple, then our algorithms can be made to run in $O(\log n \log^* n)$ on an EREW PRAM and $O(\log n)$ on a CRCW PRAM, and the number of operations is $O(n)$.

It is well-known that series parallel graphs have treewidth at most two. 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 [4] and [5]. As a side remark we note that, while the algorithms in [5] 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 [4] and [5], 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 [5], and adapted here to the setting of series parallel graphs.

This paper is organized further as follows. In Sections 2 and 3 we give some basic definitions and preliminary results. In Section 4 we give an algorithm for recognizing undirected series parallel graphs with given source and sink. This algorithm also builds an sp-tree of the input graph, if it is series parallel. Section 5 gives some results for the case that no source and sink are given and for the case that the input graph is directed. Furthermore, in this section we show how to solve many other problems on series parallel graphs.

2 Definitions

Unless stated otherwise, graphs considered are undirected, may have parallel edges but have no self-loops.

A *source-sink labeled graph* is a triple (G,s,t), where G is a graph and s and t are distinct vertices of G, called the *source* and *sink* of the graph, respectively.

The *series composition* of two or more source-sink labeled graphs is the operation which

takes $r \geq 2$ source-sink labeled graphs $(G_1, s_1, t_1), \ldots, (G_r, s_r, t_r)$ and returns a new source-sink labeled graph (G, s, t) that is obtained by taking the disjoint union of $G_1, \ldots, G_r$, identifying s_{i+1} with t_i for all i, $1 \leq i < r$, and letting $s = s_1$ and $t = t_r$. See also Figure 1.

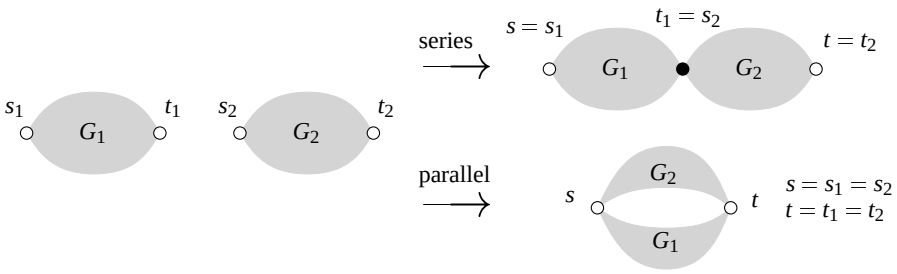

Figure 1: A series and a parallel composition of two source-sink labeled graphs.

The *parallel composition* of two or more source-sink labeled graphs is the operation which takes $r \geq 2$ source-sink labeled graphs $(G_1, s_1, t_1), \ldots, (G_r, s_r, t_r)$ and returns a new source-sink labeled graph (G, s, t) that is obtained by taking the disjoint union of $G_1, \ldots, G_r$, identifying all vertices $s_1, \ldots, s_r$ into the new source s, and identifying all vertices $t_1, \ldots t_r$ into the new sink t. See also Figure 1.

Definition 2.1 (Series Parallel Graph). *A source-sink labeled graph (G, s, t) is a* series parallel graph *if and only if one of the following holds.*

- (G, s, t) *is a* base series parallel graph*, consisting of two vertices s and t with one edge between s and t.*
- (G, s, t) *is obtained by a series or parallel composition of $r \geq 2$ series parallel graphs.*

Part I of Figure 2 shows a series parallel graph with source s and sink t. An equivalent definition which is often used only involves series and parallel compositions with two series parallel graphs. A graph G is said to be series parallel if and only if there are vertices $s, t \in V(G)$ such that (G, s, t) is a series parallel graph.

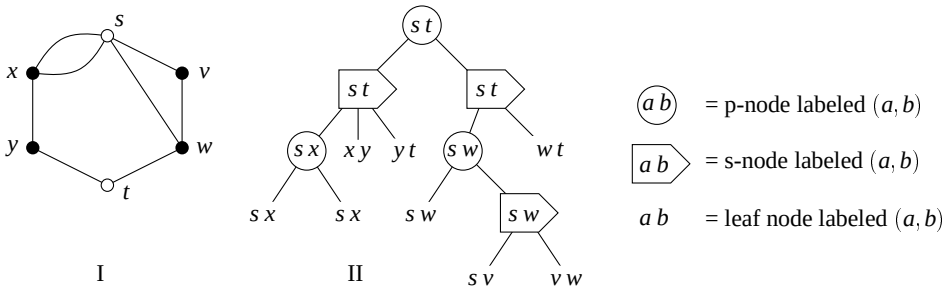

Figure 2: A series parallel graphs and its minimal sp-tree.

The 'decomposition' of a series parallel graph (G,s,t) into series and parallel compositions is expressed in an *sp-tree* T_G of the graph. An sp-tree is a rooted tree, in which each node has one of the types *p-node, s-node* and *leaf node,* and has a label. A label of a node is an ordered pair (u,v) of vertices of G. Every node of an sp-tree with label (a,b) corresponds to a series parallel graph (G',a,b), where G' is a subgraph of G. The root of the tree has label (s,t), and corresponds to the graph (G,s,t). The leaves of the tree are of type leaf node, and correspond to the base series parallel graphs that represent the edges of G: there is a one-to-one correspondence between leaves of T_G and edges $e \in E(G)$. Internal nodes are of type s-node (series node) or p-node (parallel node). The series parallel graph associated to an s-node α is the graph that is obtained by a series composition of the series parallel graphs associated to the children of α, where the order of the children gives the order in which the series composition is applied. The series parallel graph associated to a p-node β is the graph that is obtained by a parallel composition of the series parallel graphs associated to the children of β. Part II of Figure 2 shows an sp-tree of the series parallel graph given in part I.

Note that a series parallel graph can have different sp-trees. An sp-tree is called a binary sp-tree if each internal node has two children. It can be seen that any series parallel graph has a binary sp-tree. A *minimal sp-tree* of a series parallel graph (G,s,t) is an sp-tree of the graph in which p-nodes only have s-nodes and leaf nodes as children, and s-nodes only have p-nodes and leaf nodes as children. Note that the sp-tree in part II of Figure 2 is minimal. For each series parallel graph (G,s,t) there is a unique minimal sp-tree which can be obtained from any sp-tree of (G,s,t) by contracting over edges of which the end points have the same type.

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 a base series parallel graph is 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 (Treewidth). *Let $G=(V,E)$ be a graph. A* tree decomposition *TD of G is a pair (T,X), where $T=(I,F)$ is a tree, and $X=\{X_i \mid i \in I\}$ is a family of subsets of V, one for each node (vertex) of T, such that*

- $\bigcup_{i\in I} X_i = V$,
- *for every edge $\{v,w\} \in E$, there is an $i \in I$ with $v \in X_i$ and $w \in X_i$, and*
- *for all $i,j,k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

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

To be able to describe the reduction rules of our algorithm, we introduce the notion of *terminal graphs*. A terminal graph G is a triple (V,E,X) with (V,E) a graph, and $X \subseteq V$ a subset of $l \geq 0$ vertices. Vertices in X are called *terminals* or *terminal vertices,* and they are numbered from 1 to l. Vertices in $V-X$ are called *inner vertices.*

The operation $\oplus$ maps two terminal graphs G and H with the same number l of terminals to an ordinary graph $G \oplus H$, by taking the disjoint union of G and H, and then identifying the ith terminal of G with the ith terminal of H for $i = 1,\ldots,l$. See Figure 3 for an example.

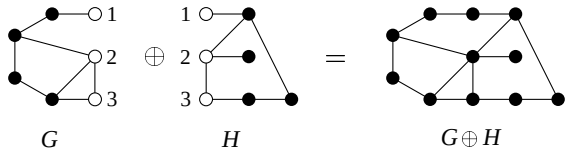

Figure 3: Example of terminal graphs and the operation $\oplus$ applied to two three-terminal graphs.

Two k-terminal graphs G_1 and G_2 are said to be *isomorphic*, if there exists an isomorphism from G_1 to G_2 which maps the ith terminal of G_1 to the ith terminal of G_2 for each i.

Definition 2.3 (Reduction Rule). *A* reduction rule *r is an ordered pair (H_1, H_2), where H_1 and H_2 are l-terminal graphs for some $l \geq 0$.*

A match *to reduction rule $r = (H_1, H_2)$ in graph G is a terminal graph G_1 which is isomorphic to H_1, such that there is a terminal graph G_2 with $G = G_1 \oplus G_2$.*

An application *of r to G is an operation that replaces G of the form $G_1 \oplus G_3$ by a graph G' of the form $G_2 \oplus G_3$, where G_1 is isomorphic to H_1 and G_2 is isomorphic to H_2. We also say that, in G, G_1 is replaced by G_2. An application of a reduction rule is also called a* reduction.

Figure 4 shows an example of a reduction rule r, and an application of r to a graph G. We usually depict a reduction rule (H_1, H_2) by the two graphs H_1 and H_2 with an arrow from H_1 to H_2. Given a reduction rule $r = (H_1, H_2)$, we call H_1 the left-hand side of r, and H_2 the right-hand side of r.

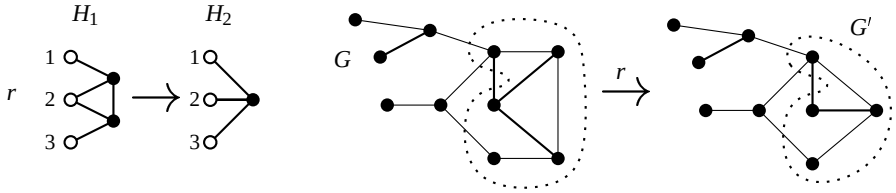

Figure 4: An example of a reduction rule $r = (H_1, H_2)$, and an application of r to a graph G, resulting in graph G'.

The notion of reductions is generalized in the natural manner to source-sink labeled 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.

3 Preliminary Results

Lemma 3.1. *Let G be a series parallel graph and let T be an sp-tree of G. If α and β are nodes of T, α is an ancestor of β, and the labels of α and β both contain a vertex v, then all nodes on the path between α and β in T contain v in their label.*

Lemma 3.2. *Let (G, s, t) is a series parallel graph with sp-tree T_G.*

1. *$(G+\{s,t\},s,t)$ is a series parallel graph, where $G+\{s,t\}$ is the graph obtained by adding an (extra) edge between s and t to G.*
2. *If there is a node α in T_G labeled with (u,v), then $(G+\{u,v\},s,t)$ is a series parallel graph.*

Proof.
1. This follows from the parallel composition of G with a one-edge series parallel graph.

2. Add between α and its parent a p-node β which has two children: node α and a leaf node representing the added edge $\{u,v\}$. The new tree is an sp-tree of $(G+\{u,v\},s,t)$. □

Lemma 3.3. *Let G be a series parallel graph, T an sp-tree of G, and $u,v \in V(G)$. The nodes in T which are labeled with (u,v) induce a (possibly empty) subtree of T.*

Lemma 3.4. *If a multigraph G is series parallel, then the treewidth of G is at most two.*

Proof. Let $T=(N,F)$ be a binary sp-tree of G. We make a tree decomposition $TD=(X,T)$ of width at most two of G from T with $X=\{X_\alpha \mid \alpha \in N\}$. For each p-node α with label (v,w), let $X_\alpha=\{v,w\}$, and for each s-node α with label (v,w) and labels of its two children (v,x) and (x,w), let $X_\alpha=\{v,w,x\}$. One can verify that (X,T) is a tree decomposition of G of treewidth at most two. □

From the construction in the proof of Lemma 3.4 it is easy to see that any binary sp-tree of G can be transformed into a tree decomposition of width at most two of G in $O(1)$ time with $O(m)$ operations on an EREW PRAM.

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 3.5. *If the treewidth of G is at most two, then G does not contain K_4 (the complete graph on four vertices) as a minor.*

Lemma 3.6. *Let (G,s,t) be a series parallel graph.*

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

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 path $(v,\ldots,x,\ldots,y,\ldots,w)$ in the graph G_β associated with node β. We use induction on 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, and has label (v,w). Let γ be the child of β on the path from α to β. If β is a p-node, then the label of γ is also (v,w). By the induction hypothesis, there is a path $(v,\ldots,x,\ldots,y,\ldots,w)$ in the graph associated with γ, and the result follows for β. 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 $\delta_j = \gamma$. For any i, $1 \leq i \leq r$, there is a path P_i from v_i to v_{i+1} in G_{δ_i} (the graph associated with δ_i). By the induction hypothesis, there is a path $P_j = (v_j,\ldots,x,\ldots,y,\ldots,v_{j+1})$ in G_{δ_j}. Concatenating $P_1,P_2,\ldots,P_r$ gives the required path of the form $(v,\ldots,x,\ldots,y,\ldots,w)$ in G_β.

2. Similar.

3. Note that there is a node with label (x,y) or a node with label (y,x). Now use part 1 of the lemma. □

Lemma 3.7. *Let (G,s,t) be series parallel and suppose there is a path $(s,\ldots,x,y,\ldots,t)$ in G. The following holds.*

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

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

2. This follows from part 1 of this lemma and Lemma 3.6. □

Lemma 3.8. *Suppose (G,s,t) is a series parallel graph with $G=(V,E)$, and let $\{x,y\} \in E$. Suppose there is a path $(s,\ldots,x,y,\ldots,t)$ in G. Let W be the set*

$$W = \{v \in V - \{x,y\} \mid \text{there is a path } (s,\ldots,x,\ldots,v,\ldots,y,\ldots,t) \text{ in } G\}.$$

Then the following holds.

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 labeled with (v,w) or (w,v), 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 p-node.*

Proof.
1. Suppose $\{v,w\} \in E$, $v \in W$, $w \notin \{x,y\}$. By Lemma 3.6, there is a path $(s,\ldots,v,w,\ldots,t)$ or there is a path $(s,\ldots,w,v,\ldots,t)$.

Suppose there is a path $(s,\ldots,v,w,\ldots,t)$. If the subpath from s to v avoids x and y, then $G+\{s,t\}$ contains K_4 as a minor, contradiction. Hence either x or y belongs to the path from s

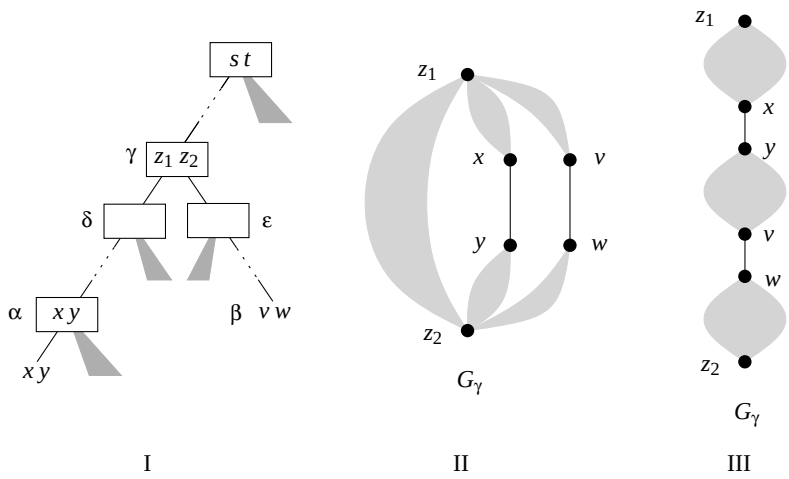

Figure 5: The sp-tree and possible graphs for the proof of Lemma 3.8

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 3.7. Hence, we have a path of the form $(s,\ldots,x,\ldots,v,w,\ldots,y,\ldots,t)$. This implies that $w \in W$.

The case in which there is a path $(s,\ldots,w,v,\ldots,t)$ is similar.

2. Note that if a node in the sp-tree of G is labeled with (v,w), then $G+\{v,w\}$ is also a series parallel graph (Lemma 3.2). Hence, the result follows from 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 3.6, there is a 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 5, part I).

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 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 5, part II shows the structure of the series parallel graph G_γ associated with node γ. The graph G_ε associated with ε contains a path $(z_1,\ldots,x,y,\ldots,z_2)$, because of Lemma 3.6, part 2. Similarly, the graph G_δ associated with node δ contains a 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 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 $V-W-\{x,y\}$, 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. Part III of

Figure 5 shows the structure of the series parallel graph G_γ. There is no path $(z_1,\ldots,v,\ldots,y)$ in G_γ, which means that any path in G which goes from x to y and contains v must be of the form $(x,\ldots,z_1,\ldots,z_2,\ldots,v,\ldots,y)$. This again means that there is an edge between a vertex in W and a vertex in $V-W-\{x,y\}$, 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. □

4 A Constructive Reduction Algorithm

In this section we give an algorithm for finding an sp-tree of a source-sink labeled graph, if it is series parallel. The algorithm is a *constructive reduction algorithm*, which consists of two phases: the first phase is the reduction phase, the second phase is the construction phase. The algorithm is based on results presented in [4, 5]. It uses a set R of reduction rules which we define later. The two phases work as follows, given a source-sink labeled graph (G,s,t).

Phase 1. The first phase consists of a number of reduction rounds which are executed subsequently. In each reduction round, a number of applications of rules from R is carried out simultaneously: if the graph is series parallel, this number is $\Omega(|E(G)|)$. In this phase, the input graph (G,s,t) is reduced to a series parallel graph consisting of one edge between vertices s and t if and only if (G,s,t) 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.

Phase 2. In the second phase, all reductions are undone in reversed order, in a number of *construction rounds*. The number of construction rounds equals the number of reduction rounds. In the first construction round, the reductions of the last reduction round of phase one are undone, in the second construction round, the reductions of the one-but-last reduction round are undone, etc., until all reductions are undone and the input graph is obtained. During the undoing of the reductions, an sp-tree of the current graph is maintained. Each time a reduction is undone, the sp-tree is 'locally' modified in such a way that it becomes an sp-tree for the new current graph. When the last construction round is finished, we obtain an sp-tree of the input graph.

In more detail, the two phases of the algorithm work as follows.

Phase 1. In phase one, the input graph is reduced to the base source-sink labeled graph if and only if the input graph is series parallel. This means that the set R of reduction rules must be *safe*, i.e. for each $r\in R$ if a graph (G',s,t) can be obtained from a graph (G,s,t) by applying r, then (G,s,t) is series parallel if and only if (G',s,t) is series parallel. The set R is given in Section 4.1, and it is shown that this set is safe.

In each reduction round in the first phase, $\Omega(|E(G)|)$ reductions are applied, if the graph is series parallel. These reductions must be *non-interfering*: no inner vertex of a subgraph that is

rewritten may occur in another subgraph that is rewritten (so the subgraphs that are rewritten may share terminals). This is to assure that the graph that results after applying all reductions of one round simultaneously is the same graph as the graph that would result if the reductions were applied subsequently in any order. Moreover, it assures that there is no concurrent reading or writing.

Finding the $\Omega(|E(G)|)$ non-interfering matches is basically done as follows. First, every edge of the current graph 'looks around' to see whether it can take part in a reduction. The set of reductions is not necessarily non-interfering, and hence a subset of non-interfering reductions is selected next. Finally, the reductions of this subset are carried out simultaneously— some bookkeeping is done such that later the reductions can be undone.

The set R of reduction rules must have the following properties to make the first step to work out correctly and fast enough.

- There is $c > 0$ such that each series parallel graph (G,s,t) with at least two edges, contains at least $c|E(G)|$ matches to rules in R. This is shown in Section 4.2.
- In each series parallel graph (G,s,t) with at least two edges, sufficiently many ($c'|E(G)|$ for some $c' > 0$) of these matches can be found in $O(1)$ time with $O(|E(G)|)$ processors. This is shown in Section 4.3.

In the second step, a subset of non-interfering reductions of all found reductions must be found. This set must be large, i.e. it must have size at least $k|E(G)|$ for some $k > 0$. This is solved in the same way as in [5]: 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 (see [5] for more details).

Finally, the set of selected reductions is carried out. Each reduction can be carried out in $O(1)$ time by a single processor.

As each reduction round reduces the number of edges with a constant fraction when the input graph is series parallel, after $O(\log m)$ reduction rounds we can conclude whether the input graph is series parallel or not, depending on whether we end up with a single edge. By using the same approach as in [5], 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.

Phase 2. The second phase builds the sp-tree, in case (G,s,t) was series parallel. This phase starts with building an sp-tree for the current graph, which is the base series parallel graph. Hence the simple sp-tree, with a single node, labeled (s,t) is build. This sp-tree is constructed in $O(1)$ time with one processor (see Section 4.4 for more details).

After that, the reduction rounds of phase one are undone in reverse order in construction rounds. During each construction round, the sp-tree is reconstructed in such a way that it becomes an sp-tree of the currect graph again. The processor that carried out the reduction in the first round will be the same processor that carries out the undoing of the reduction, and it also adapts the sp-tree locally for this undoing. How this is done is described in more detail in Section 4.4: we show that each adaptation of the sp-tree for one undoing of a reduction can be done in $O(1)$ time without interfering with other adaptations that are applied at the same time.

In this way, both a minimal and a binary sp-tree of the input graph can be obtained. Considering the fact that each undo action and local adaptation can be done in $O(1)$ time on one processor, it can be seen that phase two can be carried out in $O(\log m)$ time with $O(m)$ operations on an EREW or CRCW PRAM. This completes the description of the second phase.

With the results of Sections 4.1 – 4.4, we obtain the following result.

Theorem 4.1. *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,s,t), determine whether it is series parallel, and if so, find a minimal or binary sp-tree.*

4.1 A Safe Set of Reduction Rules

Duffin [9] has shown that a graph (G,s,t) is series parallel if and only if any sequence of applications of the *series* and the *parallel reduction rule* eventually lead to the base series parallel graph (rules 1 and 2 in Figure 6). Valdes et al. [15] have given a sequential constructive reduction algorithm for series parallel graphs, based on this reduction system, that uses $O(m)$ time.

For an efficient parallel algorithm, the series and the parallel rule are not sufficient: there are series parallel graphs which contain at most two matches to rules 1 and 2. Therefore, we introduce a larger set of reduction rules. Let R be the set of 18 reduction rules depicted in Figure 6. Note that each of the rules 3 – 18 can be applied by contracting one or two edges. These edges are marked gray in Figure 6.

In rules 3 – 18, we pose *degree constraints* on the edges between terminals: if we apply one of the rules 3 – 18 to a graph G, then in the match H that is involved in the reduction, for each edge between two terminals H, at least one of the end points of this edge has degree at most seven in G . (Note that all inner vertices of left-hand sides of rules 3 – 18 also have degree at most seven). In Figure 6, the fat edges denote the edges with a degree constraint of seven. The degree constraints are useful for proving that sufficiently many applications of the reduction rules can be found.

Hence, given a source-sink labeled graph (G,s,t), a match to a reduction rule $r = (H_1,H_2) \in R$ in (G,s,t) is a terminal graph G_1 that is isomorphic to H_1, such that

- there is a terminal graph G_2 with $G = G_1 \oplus G_2$,
- s and t are not inner vertices of G_1, and
- if r is one of the rules 3 – 18, then for each edge $e = \{u,v\} \in E(G_1)$ for which u and v are terminals of G_1, u or v has degree at most seven in G.

Safeness of rules 1 and 2, expressed in the following lemma, follows similarly as in [9]. Figures 7 and 8 illustrate how a minimal sp-tree for (G,s,t) can be transformed into one for (G',s,t) and vice versa, if $(G's,t)$ is obtained from (G,s,t) by applying rule 1 or rule 2, respectively (there are two cases for both rules).

Lemma 4.1. *If (G',s,t) is obtained from (G,s,t) by applying rule 1 or 2, then (G,s,t) is a series parallel graph if and only if (G',s,t) is a series parallel graph.*

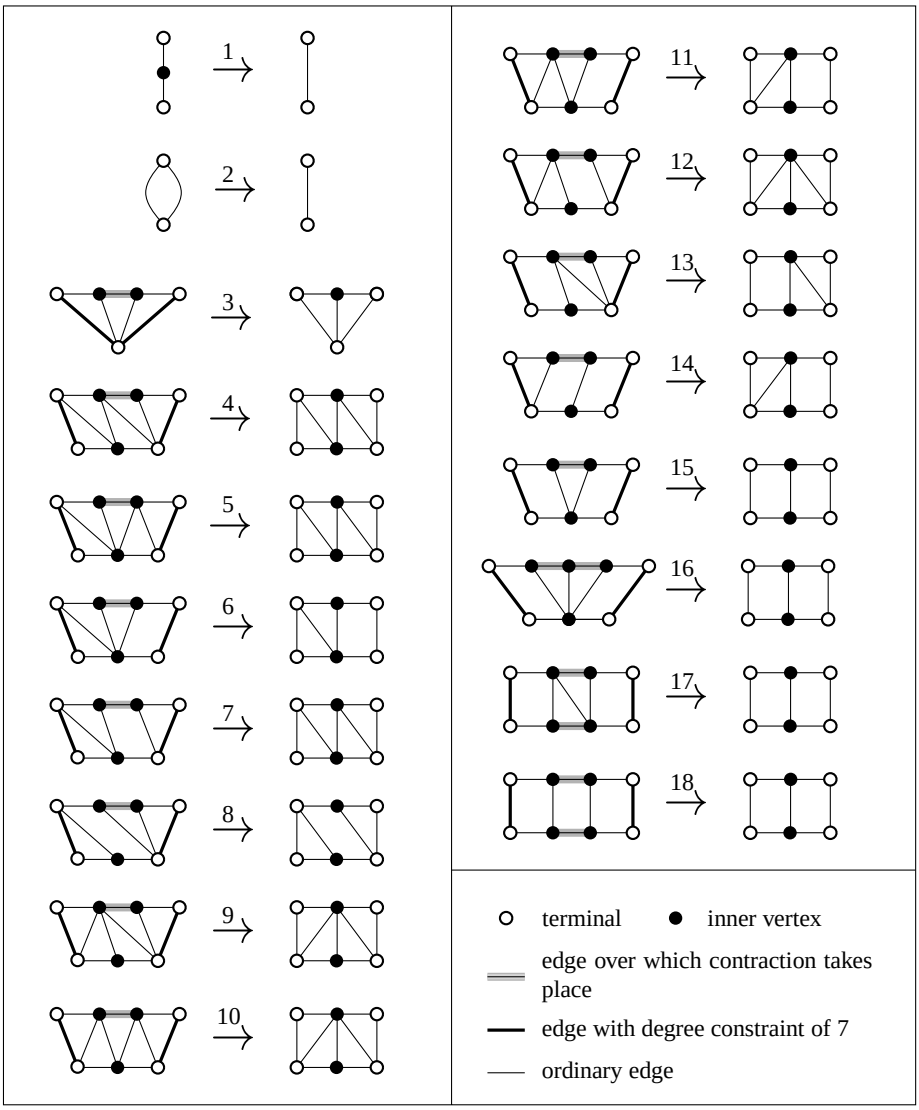

Figure 6: Reduction rules for series parallel graphs

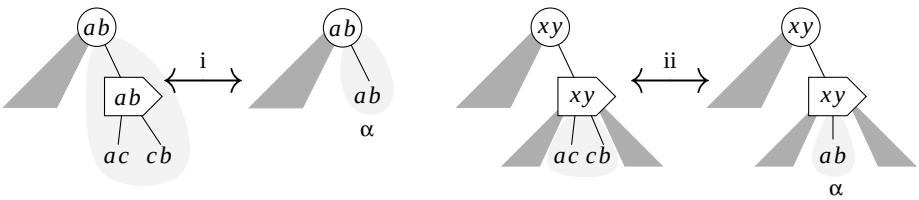

Figure 7: Transformation of sp-tree for rule 1.

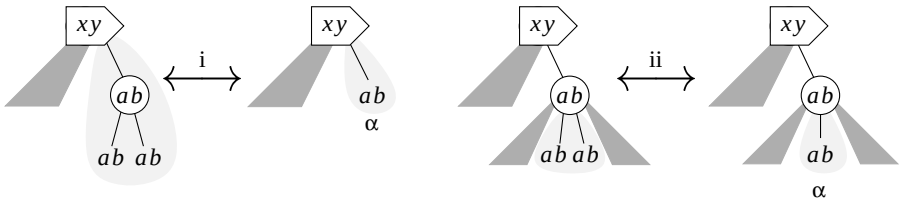

Figure 8: Transformation of sp-tree for rule 2.

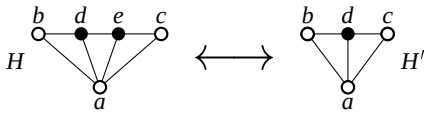

Figure 9: Matches to left-hand and right-hand sides of rule 3.

Lemma 4.2. *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). Let H be the match to rule 3, as depicted in the left-hand side of Figure 9. Suppose H is replaced by H', which is depicted in the right-hand side of Figure 9. Consider a path P from s to t that uses the edge $\{a,b\}$. We distinguish between two cases, namely the case that P visits a before b, and the case that P visits b before a.

Case 1. The path P visits a before b. We distinguish between two further cases, namely the case that P avoids e and the case that P visits e.

Case 1.1. The path P avoids vertex e. Let

$$W = \{v \in V \mid \text{there is a path } (s,\ldots,a,\ldots,v,\ldots,b,\ldots,t), \text{ and } v \text{ belongs to the same component as } e \text{ in } G[V-\{a,b\}]\}.$$

Note that $c,d,e \in W$, and hence (by part 1 of Lemma 3.8), all vertices in the component of $G[V-\{a,b\}]$ 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 3.8). Let G_α be the graph associated with α. Each vertex $v \neq a,b$ of G_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 $G[V-\{a,b\}]$ 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 $x \in V(G_\beta)$, $x \notin \{a,b\}$, then there is a path $P = (a,\ldots,x,\ldots,b)$ in G_β (Lemma 3.6). If P contains no vertex from W, then β is not an s-node. Hence P contains a vertex from W. Together with part 1 of Lemma 3.8, 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 is not 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 10, case i as a subtree. We can replace the light-gray part of this subtree by the light-gray part of the subtree shown in the right-hand side of this case and get an sp-tree of G'.

Case 1.2. The path P from s to t that uses the edge $\{a,b\}$ also uses node e. There are a two different cases, namely the case that P visits e before a, and the case that P visits e after b. In the first case, $G + \{s,t\}$ is series parallel, but contains K_4 as a minor, contradiction. In the second case, we have a path $(s,\ldots,a,e,\ldots,t)$, that does not use b. This case can be analyzed in exactly the same way as the cases above, leading to a subtree transformation as shown in Figure 10, case iii.

Case 2. 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 Figure 10, cases ii and iv.

This ends the '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. □

Lemma 4.3. *Suppose (G',s,t) is obtained from (G,s,t) by one application 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 4.2. Suppose (G,s,t) is a series parallel graph, and let T be a minimal sp-tree of (G,s,t). Let H be the match to one of the rules $4-18$ and let the terminals of H be named a, b, c and d, as shown in Figure 11 for the case that H is a match to rule 4.

Consider a path P from s to t in G 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 4.2 (define W to be the vertices of the component of $G[V - \{a,b\}]$ which contains c and d).

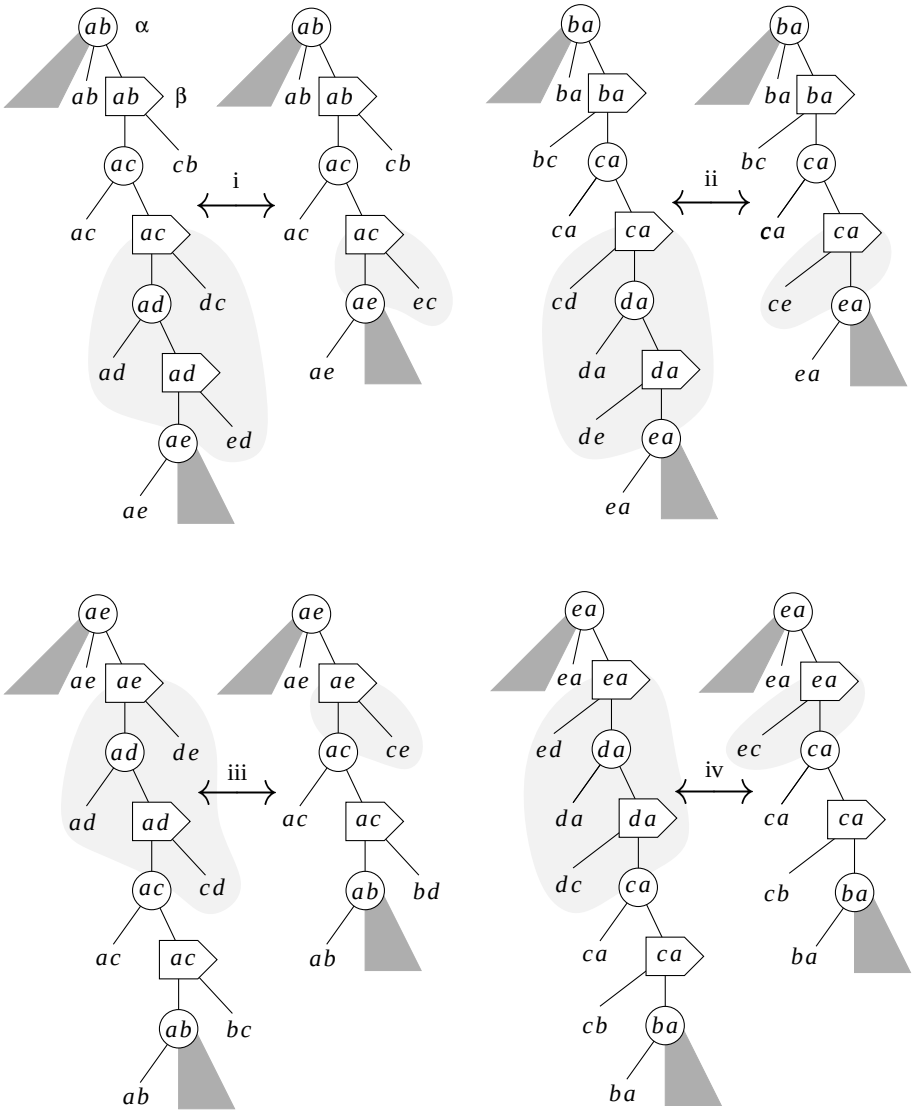

Figure 10: Transformations of subtrees for rule 3.

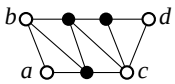

Figure 11: Match H to rule 4.

Case 2. P uses c but not d. Then either c is on the subpath $(s,\ldots,a)$ of P or c is on the subpath $(b,\ldots,t)$ 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 subpath $(s,\ldots,a)$ of P, or on the subpath $(b,\ldots,t)$ 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 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. □

We conclude with the following result.

Corollary 4.1. *The set R of reduction rules is safe for series parallel graphs*

An important consequence of the proofs of Lemmas 4.1 – 4.3 is that they 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 (see Section 4.4 for more details).

4.2 A Lower Bound on the Number of Matches

In this section we show that each series parallel graph (G,s,t) with at least two edges contains at least $\Omega(|E(G)|)$ matches to rules in R.

Lemma 4.4. *Let (G,s,t) be a series parallel graph with $|E(G)|\geq 2$. (G,s,t) contains at least $|E(G)|/139$ matches to rules 1 – 18.*

Proof. Consider the minimal sp-tree T of G. The number of leaves of T equals $|E(G)|$. We argue that the number of leaves of T is at most 139 times the number of matches. To obtain this, we distinguish the following 'classes' of leaves.

A leaf node α in T is *good* if it is a child of a p-node and has at least one sibling which is a leaf (i.e. α is child of a p-node which has at least two leaf children), or it is a child of an s-node and one of α's neighboring siblings also is a leaf node (i.e. α is child of an s-node which has at least two successive leaf children of which α is one). Note that the edges that correspond to good leaf nodes occur in matches to 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 match.

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

Now consider the other 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 matches.

Good leaves If a green s- or p-node has m good leaves, then the edges corresponding to its good leaves correspond to at least $m/2$ matches to rule 1 or 2. Hence the number of good leaves is at most twice the number of applications of reduction rules 1 and 2.

Bad leaves

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

Proof. Let α be a bad leaf. If α's parent is a p-node, then account α to its parent (which has at most one bad leaf). If α's parent is an s-node, then account α to its neighboring sibling on the right if it has one, or to its parent otherwise.

Each yellow or blue node which has a yellow or blue parent does not have any bad leaves accounted to it. Each yellow or blue node which has a branching or green parent has at most one bad leaf accounted to it, namely its neighboring sibling on the left. Let β be a yellow or blue node which has a bad leaf accounted to it. It must be the case that β has a branching or green parent. Let γ be the highest descendant of β which is green or branching. Note that there exists such a node γ. All nodes on the path from β to γ, except γ, are yellow or blue. Hence no node on this path, except β and γ, has a bad leaf accounted to it, as none of these nodes has a branching or green parent. Account the bad leaf that is accounted to β, to γ instead.

Now, each branching node has at most three bad leaves accounted to it: possibly one of its children, its neighboring sibling on the left, and one leaf first accounted to a yellow or blue node. Each green node has at most two bad leaves accounted to it: again possibly one from a yellow or blue node, and as green s-nodes have no bad leaf children and green p-nodes have no bad siblings, at most one other bad leaf. □

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

Proof. Construct a 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 in any tree, the number of internal nodes 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. □

Claims 4.1 and 4.2 show that the number of bad leaves is at most $3+2=5$ times the number of green nodes. In each green node, there is a match to rule 1 or 2 in two of the edges corresponding to its good leaves. Hence, the number of bad leaves is at most five times the number of matches to rules 1 and 2.

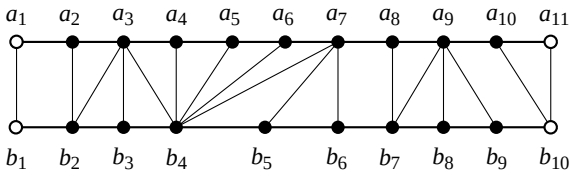

Figure 12: Subgraph of G corresponding to a path of 33 yellow or blue nodes in the sp-tree, of which the highest one is a p-node with label (a_1, b_1), and the lowest one is a p-node with label (a_{11}, b_{10}). Only a_1, b_1, a_{11} and b_{10} may be incident with edges outside the subgraph.

Leaves 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. 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 matches to 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 p-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-neighboring 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 a common edge. As no s-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 12 for an example.

Claim 4.3. In a cycle-sequence of G that consists of 16 cycles, there is a match to one of the rules 3 – 18.

Proof. We omit the full proof here: a long and tedious case analysis shows this fact. Full details are given in [8]. □

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 matches to rules 3 – 18 is at least $1/34$ times the number of yellow nodes. Since each yellow node has at most two leaf children, we have that the number of leaf children of yellow nodes is at most $2 \cdot 34 = 68$ times the number of matches to rules 3 – 18.

The total number of leaves in T is now at most $2+5+132=139$ times the number of matches to rules 1 and 2 plus 68 times the number of matches to rules 3 – 18. Hence the number of leaves in T is at most 139 times the number of matches in R. This completes the proof. □

4.3 A Lower Bound on the Number of Enabled Matches

In this section we show that, in a series parallel graph (G,s,t) with at least two edges, we can find $c|E(G)|$ matches to rules in R in $O(1)$ time with $O(|E(G)|)$ processors.

The finding of the matches is done as follows. Every edge of the current graph 'looks around' to see whether it can take part in a reduction. Given an edge e it can easily be checked whether e can occur in an application of one of the rules 1 or 3 – 18: follow all paths of length at most eight from e which visit only vertices of degree at most eight (except for the last vertex of a path). This can be done in $O(1)$ time per node per edge. In this way, all possible choices for applications of these rules are found. However, for rule 2, probably not all possible applications can be found in this way. Instead, for rule 2, every edge $e=\{u,v\}$ searches in the adjacency lists of u and v for all edges that have distance at most ten to e in this list. Edge e proposes an application of rule 2 if one of the edges it found also has end points u and v. Thus, these rule applications can also be found in $O(1)$ time. (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 4.5. *If $G=(V,E)$ is a simple series parallel graph, then $|E|\leq 2|V|$.*

Proof. Follows from the fact that each series parallel graph has treewidth at most two, and a simple graph G of treewidth at most k has at most $k|V(G)|$ edges for any $k\geq 1$. □

Let G be a graph given by some adjacency list representation. An edge in G is called *bad* if it has a parallel edge, but no parallel edge is found in the procedure above.

Lemma 4.6. *Let G be a graph of treewidth at most two given by some adjacency list representation. There are at most $|E(G)|/5$ bad edges in G.*

Proof. Consider a tree decomposition (T,X) of G of width at most two with $T=(I,F)$ and $X=\{X_i \mid i\in I\}$, 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 $v\in X_{r_v}$. Let $e\in E$ with end points v and w. There is a node containing v and w, hence either $r_v=r_w$, or r_v is an ancestor of r_w, or r_w is an ancestor of r_v.

For every bad edge between v and 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 e between v and w is associated with v. Then X_{r_v} must contain both v and w. It follows that there are at most $|X_{r_v}|-1\leq 2$ different vertices u for which bad edges between v and u can be associated with v (namely, the vertices in $X_{r_v}-\{v\}$). For each such u, each 20 successive positions in the (cyclic) adjacency list of v can contain at most one bad edge between u and v, hence there are at most $\deg(v)/20$ bad edges between v and u that are associated with v, and hence in total, at most

$\deg(v)/10$ bad edges are associated with v. The stated bound is derived by taking the sum over all vertices. □

As each series parallel graph has treewidth at most two, it follows that each series parallel graph (G,s,t) has at most $|E(G)|/5$ bad edges.

Lemma 4.7. *There is a constant $c > 0$ for which each series parallel graph (G,s,t) with at least two edges contains at least $c|E(G)|$ enabled matches.*

Proof. Let $n = |V(G)|$ and $m = |E(G)|$. We distinguish two cases, namely that case that $m \geq 4n$ and the case that $m < 4n$. If $m \geq 4n$, then there are at least $m - 2n$ edges that are parallel to another edge, of which at most $m/5$ are bad. Hence, there are at least $4/5m - 2n \geq 4/5m - 1/2m = 3m/10$ edges e for which there is a parallel edge which has distance at most 10 to e in the adjacency list of one of the end points of e. This implies that there are at least $3m/20$ enabled matches to rule 2 in (G,s,t).

Suppose $m < 4n$. Let G' be the simple graph underlying G, i.e., G' is obtained from G by all second and further occurrences of parallel edges. Note that (G',s,t) is a series parallel graph, and G' has at least $n-1$ edges. If G' has one edge, then G consists of two vertices with $m \leq 8$ parallel edges, and hence G contains at least one enabled match to rule 2, so at least $m/8$ enabled matches.

Suppose G' has at least two edges. By Lemma 4.4 there are at least $(n-1)/139 \geq n/278$ matches to rules 1 and $3-18$ in (G',s,t). As each of the matches to rules 1 and $3-18$ is enabled in G', this implies that (G',s,t) has at least $n/278$ enabled matches. For each match in this set, there are two possibilities: either it is also an enabled match in G, or it is disturbed by the addition of one or more parallel edges. We will call a match of the first type a *non-disturbed* match, and a match of the last type a *disturbed* match. We now show that the number of disturbed matches is at most k times the number of matches to rule 2 in G, for some positive integer k.

Consider a disturbed match H. There are two cases: either an inner vertex v of H is incident with parallel edges, or a terminal vertex v which has degree at most seven in G' has degree more than seven in G (and hence is incident with parallel edges). In both cases, the vertex v has degree at most seven in G', and hence in G, there is an enabled match to rule 2 which contains vertex v: any sublist of length 20 of the adjacency list contains at least two edges with the same end points, as there at most seven different sets of end points possible.

Account each disturbed match in G' to an enabled match to rule 2 in G which contains a vertex of degree at most seven of the disturbed match. It can be seen that each vertex of degree at most seven in G is contained in at most k matches for some constant k. Hence each enabled match to rule 2 in G has at most $2k$ disturbed matches accounted to it.

Consider the number of enabled matches in G. This number is at least the number of non-disturbed matches plus the number of enabled matches to rule 2 in G, which is at least the number of non-disturbed matches plus $1/(2k)$ times the number of disturbed matches. Hence the number of enabled matches in G is at least $1/(2k)$ times the number of enabled matches in G'. This latter number is at least $n/278$, and hence there are at least $n/(556k)$ enabled matches. As $m < 4n$, this means that there are at least $m/(2224k)$ enabled matches in G. □

Note that the constant c in Lemma 4.7 is quite bad. However, the bound we have derived can probably be tightened by using more detailed estimates.

4.4 The Construction Phase

In this section, we show in more detail how a minimal sp-tree of the input graph (G,s,t) is build.

The sp-tree is represented as follows. We make a list of all nodes in the sp-tree. Each node is marked with its label and its type (s-node, p-node or leaf node), each node has a pointer to its left-most and its right-most child, to its parent, and to its neighboring siblings on the left-hand and the right-hand side (if one of these nodes does not exist, the pointer is nil). Furthermore, each leaf node is marked with the type of its parent, and we keep a pointer from each edge in the graph to the corresponding leaf node in the sp-tree.

We start with the simple sp-tree, with a single node, labeled (s,t). It is easy to see that this sp-tree can be constructed in $O(1)$ time with one processor, and that it is an sp-tree of the current graph.

In each construction round, the sp-tree is reconstructed: each processor that carried out a reduction in the first phase, undoes this reduction in this phase, and adapts the sp-tree locally for this undoing. For these adaptations we use the constructions from the proofs of Lemmas 4.1 – 4.3 (see also Figures 7 – 10). We show that each adaptation of the sp-tree for one undoing of a reduction can be done in $O(1)$ time without interfering with other adaptations that are applied at the same time.

Suppose a reduction rule $r = (H_1, H_2) \in R$ has to be undone, and let G_1 and G_2 be terminal graphs such that G_1 and H_1 are isomorphic and G_2 and H_2 are isomorphic, and G_2 is the match in the current graph that has to be replaced by G_1. The adaptation of the sp-tree for this undoing is done as follows. First the local structure of the sp-tree is found, i.e. the structure of the part of the sp-tree that contains edges in G_2 is found. For rules 1 and 2, the different forms are the right-hand sides of cases i and ii in Figures 7 and 8, respectively. For rule 3, the different forms are the right-hand sides of cases i, ii, iii and iv in Figure 10. The parts of the sp-tree that are marked light-gray are the parts that must be modified.

The local structure is found as follows. An edge e of G_2 is taken which is not an edge between two terminals in the case of rules 3 – 18 (for rules 1 and 2, the only possibility is the edge $\{a,b\}$, for rule 3, edge $\{c,e\}$ is the best edge to take, as this edge will be removed). Look at the corresponding leaf node in T. For rules 1 and 2, check the type of its parent node, and for rule 3 – 18, search the 'neighborhood' of this leaf node in T which is involved in the modification (for rule 3, this is the light-gray part in the right-hand side of cases i, ii, iii and iv in Figure 10). The leaf node can be found in constant time without interfering with any other constructions. For rules 1 and 2, it is clear that we can check the type of its parent in constant time without interfering with other constructions performed at the same time, as each leaf node is marked with the type of its parent. For rule 3, we can see from Figure 10 that the structure of the neighborhood can be determined in $O(1)$ time without interfering with other constructions, as no other construction involves any of the nodes of the light-gray part of the sp-tree. For rules 4 – 18, the cases are similar to the cases of rule 3, and the structure can also be found in $O(1)$ time without interference.

After the local structure of the sp-tree is found, this part of the sp-tree is replaced by a new part. The structure of this new part depends on the structure of the old part. For rules 1, 2 and 3, these new parts are the parts in left-hand sides of the cases in Figures 7, 8 and 10 that are marked light-gray. For rules 4 – 18, a similar approach as for rule 3 can be taken. For rules 3 – 18, it is easy to see that the modification can be done in $O(1)$ time without interference. For rules 1 and 2, case i is also easy (see Figures 7 and 8: node α gets a different type, and gets two leaf children). In case ii, the modification needs more care, as the neighboring siblings of node α may be leaf nodes that are involved in another rule 1 or 2 reduction at the same time. Hence we have to ensure that the corresponding executions do not read from or write to the same memory location at the same time. This is done by inserting the new leaf node as a right-hand sibling of the leaf node for $\{a,b\}$, which can be done in $O(1)$ time without interference.

This completes the description of the construction phase of the algorithm in the case a minimal sp-tree is build. In the same way, we can build a binary sp-tree of the graph.

5 Additional Results for Series Parallel Graphs

In this section we show that the algorithm presented in Section 4 can also be used to solve the 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, and want to determine whether G is series parallel with a proper choice of the source and sink. We solve this problem by first computing a source and a sink and then solving the problem with this source and sink. He [11] and Eppstein [10] have shown (using results from Duffin [9]) that this problem reduces in a direct way to the problem with specified vertices, as the following result holds.

Lemma 5.1 [11, 10]. *Let $G = (V,E)$ be a graph. If G is series parallel then the following holds.*

1. *If G is not biconnected, then the blocks of G form a path: each cut vertex of G is in exactly two blocks, all blocks have at most two cut vertices, and there are exactly two blocks which contain one cut vertex.*
2. *The graph (G,s,t) is series parallel if s and t are vertices of G chosen as follows.*
   (a) *If G is biconnected, then s and t are adjacent.*
   (b) *If G is not biconnected, then let B_1 and B_2 be the blocks of G which contain one cut vertex, and let c_1 and c_2 denote these cut vertices. Source s is a vertex of B_1 which is adjacent to c_1, and sink t is a vertex of B_2 which is adjacent to c_2.*

We briefly describe how s and t can be found such that they satisfy conditions 2a and 2b of Lemma 5.1. Therefore, we apply a result of [4], which says that any graph problem which can be defined in *monadic second order logic* for graphs, can be solved on graphs of bounded treewidth in time $O(\log n \log^* n)$ with $O(n)$ operations on an EREW PRAM, and in $O(\log n)$ time with $O(n)$ operations on a CRCW PRAM. However, this result does not apply

to multigraphs, and our input graph is a multigraph. Therefore, we make a new, simple graph $G' = (V', E')$ from the multigraph G as follows.

$$V' = V(G) + E(G)$$
$$E' = \{\{v, e\} \mid v \in V(G) \wedge e \in E(G) \wedge v \text{ is incident with } e\}$$

We make a labeling of the vertices in G': each vertex originating from $V(G)$ is labeled vertex, and each vertex originating from $E(G)$ is labeled edge. It is easy to see that the resulting graph is a simple graph and has $n+m$ vertices and $2m$ edges, and furthermore, if G is series parallel, then G' is series parallel. The transformation can be performed in $O(1)$ time with $O(n+m)$ operations.

It can be seen that the characterization of s and t as given in Lemma 5.1 can be translated to a characterization of s and t in the modified, simple graph G', and this characterization can be formulated in monadic second order logic for graphs (using techniques from e.g., [6]). Hence, it is possible (using techniques of [5, 4]) to find s and t 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.)

If the input graph is a source-sink labeled directed graph (G, s, t), then one can use the modification, described in [10]: solve the problem on the underlying undirected graph, then orient the edges with help of the minimal sp-tree (there is at most one possible orientation for which the directed graph is series parallel), and check if this orientation corresponds to the original graph.

If the input graph is directed, and no source and sink are specified, then there must be exactly one vertex with indegree zero and one with outdegree zero, otherwise, the graph is not series parallel. Let the source be this first vertex, and the sink the latter vertex, and solve the problem for the graph with this source and sink. Note that these vertices can be found in $O(\log m)$ time with $O(m)$ operations on an EREW PRAM.

Theorem 5.1. *Each of the following problems can be solved with $O(m)$ operations, in $O(\log m \log^* m)$ time on an EREW PRAM, and $O(\log m)$ time on a CRCW PRAM.*

1. *Given a graph G, determine if there are $s, t \in V(G)$ for which (G, s, t) is series parallel, and if so, find an sp-tree of G.*
2. *Given a directed source-sink labeled graph (G, s, t), determine whether (G, s, t) is series parallel, and if so, find an sp-tree of (G, s, t).*
3. *Given a directed graph G, determine if there are $s, t \in V(G)$ for which (G, s, t) is series parallel, and if so, find an sp-tree of G.*

If the input graph is simple, then we can make the algorithms to run in $O(\log n \log^* n)$ time on an EREW PRAM and $O(\log n)$ time on a CRCW PRAM, both with $O(n)$ operations and space. This can be done by making use of the fact that a simple series parallel graph has at most $2n$ edges, and the fact that $2n$ edges can be counted in $O(\log n)$ time with $O(n)$ operations: in a preprocessing step, start counting the number of edges of the graph, but we do at most $O(\log n)$

steps of this counting with $O(n)$ operations. If, after these steps, the edges are counted and $m \leq 2n$, then go on with the rest of the algorithm. Otherwise, conclude that the graph is not series parallel and return false.

Many problems can be solved in $O(\log p)$ time, and $O(p)$ operations and space, when the input graph is given together with a tree decomposition of bounded treewidth consisting of p nodes. 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 [2, 7, 5].

Since series parallel graphs have treewidth at most two, we can solve these problems efficiently on series parallel graphs, if a tree decomposition of small width is given. A binary sp-tree of a series parallel graph can be transformed into a tree decomposition of width at most two in constant time, by using the construction of Lemma 3.4. Hence we have the following result.

Corollary 5.1. *The following problem can be solved in $O(\log m \log^* m)$ time, $O(m)$ operations, and $O(m)$ space on an EREW PRAM, and in $O(\log m)$ time, $O(m)$ operations and $O(m)$ space on a CRCW PRAM: given a series parallel graph G, find a tree decomposition of width at most two of G.*

The resulting tree decomposition has $O(m)$ nodes. Hence we can solve the problems described above in $O(\log m)$ time with $O(m)$ operations given this tree decomposition.

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] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340, 1991.

[3] 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.

[4] H. L. Bodlaender and B. de Fluiter. Reduction algorithms for constructing solutions in graphs with small treewidth. In J.-Y. Cai and C. K. Wong, editors, *Proceedings 2nd Annual International Conference on Computing and Combinatorics, COCOON'96*, pages 199–208. Springer Verlag, Lecture Notes in Computer Science, vol. 1090, 1996.

[5] 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.

[6] 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.

[7] B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[8] B. de Fluiter. *Algorithms for Graphs of Small Treewidth.* PhD thesis, Utrecht University, 1997.

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

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

[11] X. He. An improved algorithm for the planar 3-cut problem. *J. Algorithms*, 12:23–37, 1991.

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

[13] 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.

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

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