Parallel Algorithms for Treewidth Two[*]

Babette de Fluiter
Centre for Quantitative Methods
P.O. Box 414, 5600 AK Eindhoven, the Netherlands
e-mail: deFluiter@cqm.nl

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

Abstract

In this paper we present a parallel algorithm that decides whether a graph G has treewidth at most two, and if so, construct a tree decomposition or path decomposition of minimum width of G. The algorithm uses $O(n)$ operations and $O(\log n \log^* n)$ time on an EREW PRAM, or $O(\log n)$ time on a CRCW PRAM. The algorithm makes use of the resemblance between series parallel graphs and partial two-trees. It is a (non-trivial) extension of the parallel algorithm for series parallel graphs that is presented in [8].

1 Introduction

In this paper we consider the problem of finding a tree decomposition of width at most two of a graph, if one exists.

Many important graph classes have bounded treewidth. Given a tree decomposition of bounded width of a graph, many (even NP-hard) problems can be solved sequentially in linear time, and in parallel in $O(\log n)$ time with $O(n)$ operations on an EREW PRAM, where n denotes the number of vertices of the graph (see e.g. [5, 9]) (the number of operations that an algorithm uses is the product of the number of processors and the time it uses). Therefore, the problem of finding a tree decomposition of bounded width of a graph is well studied.

Sequentially, there exist linear time algorithms for each fixed k that, when given a graph G, decide whether the treewidth of G is at most k, and if so, build a tree decomposition of minimum width for G. Practical algorithms exist for $k = 1$, 2, 3, and 4 [4, 16, 17]; in [6], linear time algorithms are given for each fixed k.

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

The best known parallel algorithm for *recognizing* graphs of treewidth at most k was found by Bodlaender and Hagerup [9]. It uses $O(n)$ operations, with $O(\log n)$ time on a CRCW PRAM or $O(\log n \log^* n)$ time on an EREW PRAM. They also gave a parallel algorithm for *building* a tree decomposition of width at most k, which uses $O(n)$ operations and $O(\log^2 n)$ time on a CRCW or EREW PRAM. Related, earlier results can be found e.g. in [14, 15].

For treewidth one there is a more efficient algorithm than the one of [9]. A connected simple graph has treewidth one if and only if it is a tree, and a tree can be recognized by using a tree contraction algorithm. This takes $O(\log n)$ time with $O(n)$ operations on an EREW PRAM [1]. One can easily construct a tree decomposition of a tree in $O(1)$ time with $O(n)$ operations on an EREW PRAM. The algorithm can be modified such that it can be used on input graphs which are not necessarily connected (see also Section 4).

In this paper, we improve on the algorithm of [9] for treewidth two. Our algorithm constructs a tree decomposition of width at most two of a graph, if the graph has treewidth at most two. It uses $O(n)$ operations, $O(\log n)$ time on a CRCW PRAM and $O(\log n \log^* n)$ time on an EREW PRAM. We also obtain an algorithm solving the problem on multigraphs, which uses $O(n+m)$ operations with $O(\log(n+m)\log^*(n+m))$ time on an EREW PRAM and $O(\log(n+m))$ time on a CRCW PRAM. From these results and a result from [9] we immediately obtain parallel algorithms for the problem of finding a path decomposition of width at most two of a graph, if it has pathwidth at most two, both for the case of simple graphs and multigraphs. These algorithms run in the same time and resource bounds as the algorithms for treewidth two.

A central technique in this paper is *graph reduction*, introduced in [2]. In [7] and [9] it is shown how the technique can be used to obtain parallel algorithms for graphs of bounded treewidth. In [8] this technique is used for checking whether a given graph is series parallel, and if so, finding a decomposition of the graph in series and parallel compositions. Our algorithm for treewidth two uses the resemblance between series parallel graphs and graphs of treewidth two: we show that a graph has treewidth at most two if and only if its biconnected components are series parallel. We modify the reduction algorithm that is presented in [8] for recognizing series parallel graphs in order to obtain an algorithm for graphs of treewidth at most two: we add extra reduction rules and show how a tree decomposition of width at most two is constructed. (Also, the counting arguments needed for showing the time bounds for this algorithm are different from those of the algorithm in [8].) It is interesting to note that the sequential algorithm to recognize graphs of treewidth two [4, 16] is also based upon a (much smaller) set of graph reductions.

This paper is organized as follows. In Section 2 we start with definitions and preliminary results. In Section 3 we give a reduction algorithm which checks whether a given graph has treewidth at most two, and if so, finds a tree decomposition of width at most two of the graph. We give this algorithm for a special type of input graph, namely a *connected* B*-labeled multigraph*. Finally in Section 4, we show how this algorithm can be used for general multigraphs and simple graphs, and we give some additional results.

2 Preliminaries

Unless stated otherwise, graphs considered are undirected, may have parallel edges but have no self-loops. Graphs without parallel edges are called simple graphs.

A biconnected component of a graph is also called a *block*. A block is *trivial* if it consists of one edge. All other blocks are *non-trivial*.

Definition 2.1 (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 $\max_{i \in I} |X_i| \Leftrightarrow 1$. The* treewidth *of a graph G is the minimum width over all possible tree decompositions of G.*

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 2.1. *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.*

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

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

A source-sink labeled graph is a *series parallel graph*, if and only if it is a single edge, or it is obtained by a series or parallel composition of $r \geq 2$ series parallel graphs. A graph G is series parallel, if there are vertices s, t, such that the source-sink labeled graph (G,s,t) is series parallel.

Lemma 2.2. *A graph G has treewidth at most two if and only if each block of G is series parallel.*

Proof. Suppose G has treewidth at most two. Let G' be a block of G (G' has treewidth at most two). We show by induction on $|V(G')| + |E(G')|$ that G' is series parallel.

If $|V(G')| \leq 3$, then it clearly holds. Suppose $|V(G')| > 3$, note that $|E(G')| \geq |V(G')|$. If G' contains parallel edges, then apply a parallel reduction on G'. The graph obtained this way has treewidth two and is biconnected. By the induction hypothesis, it is series parallel. This implies that G' is series parallel.

Suppose G' does not contain any parallel edges. Let $TD = (T,X)$ be a tree decomposition of width two of G' with $T = (I,F)$ and $X = \{X_i \mid i \in I\}$. Modify TD by repeating the following as often as possible. For each $i \in I$, if i has exactly one neighbor $j \in I$, and $X_i \subseteq X_j$, then remove X_i. Note that TD is a tree decomposition of width two of G', and it has at least two nodes. Let $i \in I$ such that i has exactly one neighbor $j \in I$ in T. There is a $v \in X_i$ such that $v \notin X_j$.

Let $v \in X_i$ such that $v \notin X_j$. Vertex v must have degree two in G', and both v's neighbors are contained in X_i. Apply the series reduction on v and its neighbors u and w. This gives the graph $G'' = (V(G') \Leftrightarrow \{v\}, E(G') + \{u,w\})$. Graph G'' has treewidth two, since the tree decomposition obtained from (T,X) by removing vertex v from node X_i is a tree decomposition of width two of G''. Furthermore, G'' is biconnected. By the induction hypothesis, G'' is a series parallel graph, and thus G' is also series parallel.

Now suppose each block of G is series parallel. By Lemma 3.4 of [8], each block of G has treewidth at most two, and hence the treewidth of G is at most two. $\square$

We briefly describe the notion of *terminal graphs* and *reduction rules* here. For a more detailed description, see Section 2 of [8].

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*, and they are numbered from 1 to l. Vertices in $V \Leftrightarrow X$ 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 $i = 1, \ldots, l$.

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.

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*.

A reduction rule (H_1,H_2) is *safe* for a class of graphs $\mathcal{G}$, if for all terminal graphs H_3 with the same number of terminals as H_1 (and H_2): $H_1 \oplus H_3 \in \mathcal{G} \Leftrightarrow H_2 \oplus H_3 \in \mathcal{G}$.

Let $G = (V,E)$ be a graph. A *bridge* of G is an edge $e \in E$ for which the graph $(V, E \Leftrightarrow \{e\})$ has more connected components than G. In order to make the set of reduction rules conveniently small, we put a labeling on the edges of a graph: each edge in a graph is either labeled with label B, or it is not labeled (the label B stands for 'bridge'). We call such a graph a B-*labeled graph*. We extend the notion of treewidth at most two for graphs to treewidth at most two for B-labeled graphs.

Definition 2.2. *Let $G = (V,E)$ be a connected* B*-labeled graph. Let G' be the underlying unlabeled graph. The graph G has* treewidth at most two *if and only if G' has treewidth at most two and for each edge $e \in E$, if e has label* B*, then e is a bridge of G.*

A tree decomposition of width at most two *of G is a tree decomposition $TD = (T,X)$ of width at most two of G' with $T = (I,F)$ and $X = \{X_i \mid i \in I\}$, such that for each edge e with* *label* B *and end points u and v, there is a node $i \in I$ with $X_i = \{u,v\}$ such that there is no component in $T[I \Leftrightarrow \{i\}]$ which contains both u and v.*

We can easily prove by induction that a B-labeled graph G has treewidth at most two if and only if there is a tree decomposition of width at most two of G.

Note that an edge in a graph is a bridge if and only if the edge is a (trivial) block. Hence we can derive the following from Lemma 2.2.

Corollary 2.1. *Let G be a connected* B*-labeled graph. G has treewidth at most two if and only if each non-trivial block of G has no labeled edges and is series parallel.*

We use B-labeled terminal graphs instead of unlabeled ones: a B-labeled terminal graph is a terminal graph of which edges may have label B. Two B-labeled terminal graphs G_1 and G_2 are isomorphic if there is an isomorphism from the underlying unlabeled terminal graph of G_1 to the underlying unlabeled terminal graph of G_2, such that labeled edges in G_1 are mapped to labeled edges in G_2 and unlabeled edges in G_1 are mapped to unlabeled edges in G_2.

Reduction rules consist of pairs of B-labeled terminal graphs instead of ordinary terminal graphs.

The following result on trees is used in Section 3.2.

Lemma 2.3. *Let H be a tree. Let $l(H)$ denote the number of leaves of H, and let $nr(H)$ denote the sum of the degrees of all vertices of degree at least three. Then $nr(H) \leq 3l(H)$.*

Proof. We prove this by induction on the number n of vertices of H. If $n \leq 2$, then clearly $nr(H) \leq 3l(H)$.

Suppose $n > 2$. Let v be a leaf of H, and let w be the only neighbor of v. Let d denote the degree of w in H and note that $d \geq 2$. Furthermore, let $H' = H[V \Leftrightarrow \{v\}]$. By the induction hypothesis, $nr(H') \leq 3l(H')$. If $d = 2$, then $l(H) = l(H')$ and $nr(H) = nr(H')$, so $nr(H) \leq 3l(H)$. If $d = 3$, then $l(H) = l(H') + 1$ and $nr(H) = nr(H') + 3$, and thus $nr(H) \leq 3l(H)$. If $d \geq 4$, then $l(H) = l(H') + 1$ and $nr(H) = nr(H') + 1$, and hence also $nr(H) \leq 3l(H)$. □

3 A Constructive Reduction Algorithm

In this section we give an algorithm for finding a tree decomposition of width at most two of a connected B-labeled graph, if one exists. In Section 4 it is shown how this algorithm can be used for graphs which may be unconnected or for simple graphs.

The structure of the algorithm for connected B-labeled graphs is the same as the algorithm for series parallel graphs as presented in Section 4 of [8]: it is a *constructive reduction algorithm*. The algorithm consists of two phases: the first phase is the reduction phase, in which the input graph is reduced to a graph consisting of one vertex, if it has treewidth at most two. The second phase is the construction phase, in which the reductions are undone in reverse order, and a tree decomposition of the graph is constructed during the undoing of the reductions. The algorithm uses a set R_{tw} of reduction rules, which we define later. We briefly describe the basic idea of the two phases (for more details, see Section 4 of [8]). Suppose a B-labeled graph G is given.

Phase 1. The first phase consists of a number of reduction rounds. In each reduction round, a number of applications of rules from $\mathcal{R}_{\text{tw}}$ is carried out simultaneously. In this phase, the input graph G is reduced to a single vertex if and only if G has treewidth at most two. If G has treewidth more than two, then the algorithms stops. Otherwise, it proceeds with the second phase.

The set $\mathcal{R}_{\text{tw}}$ must be *safe*: for each $r \in \mathcal{R}_{\text{tw}}$, if a B-labeled graph G' can be obtained from a B-labeled graph G by applying r, then G has treewidth at most two if and only if G' has treewidth at most two. In Section 3.1, we give the set $\mathcal{R}_{\text{tw}}$ of reduction rules, and we show that it is safe.

In each reduction round, $\Omega(|E(G)|)$ reductions are applied, if the graph has treewidth at most two. These reductions must be non-interfering: no inner vertex of a subgraph that is rewritten may occur in another subgraph that is rewritten (see also Section 4 of [8]).

Finding the $\Omega(|E(G)|)$ non-interfering matches is done as follows. First, a set of $\Omega(|E(G)|)$ matches is found. Next a subset of $\Omega(|E(G)|)$ non-interfering reductions is selected from this set. This is done similar as in [8] with a technique from [9]. Finally, these non-interfering reductions are carried out simultaneously.

The second and third step are done in the same way as for series parallel graphs (see also [9]): each reduction in the third round is carried out by a single processor in $O(1)$ time. To be able to find $\Omega(|E(G)|)$ matches sufficiently fast, we need the following two properties of $\mathcal{R}_{\text{tw}}$.

- There is $c > 0$ such that each connected B-labeled graph G with at least one edge contains at least $c|E(G)|$ matches to rules in $\mathcal{R}_{\text{tw}}$. This is shown in Section 3.2.
- In each connected B-labeled graph G with at least one edge, 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 3.3.

Phase one can be carried out in $O(\log m \log^* m)$ time on an EREW PRAM and in $O(\log m)$ time on a CRCW PRAM, both with $O(m)$ operations (see [8] and [9] for more details).

Phase 2. In the second phase, all reductions are undone in reversed order in the construction rounds. During the undoing of the reductions, a tree decomposition of width at most two of the current graph is maintained. Each time a reduction is undone, the tree decomposition is 'locally' modified. When the last construction round is finished, we obtain a tree decomposition of the input graph.

The undoing of a reduction is carried out by the same processor which carried out the reduction in phase one. This processor also locally modifies the tree decomposition for this reduction. Each undo action of a reduction, including the reconstruction of the tree decomposition, is done in $O(1)$ time by one processor. This implies the phase two can be done in $O(\log m)$ time with $O(m)$ operations on an EREW or CRCW PRAM. In Section 3.4 we describe in more detail how the construction of the tree decomposition is done.

Phase two can be carried out in $O(\log m)$ time with $O(m)$ operations on a CRCW or EREW PRAM.

Together with the results of Sections 3.1 – 3.4 and the results described in Section 4 of [8], we obtain the following theorem.

Theorem 3.1. *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 connected* B*-labeled graph G, determine whether it is has treewidth at most two, and if so, find a tree decomposition of width at most two of G.*

3.1 A Safe Set of Reduction Rules

The set $\mathcal{R}_{\text{tw}}$ of reduction rules for treewidth at most two is depicted in Figure 1. It is an extension of the set of reduction rules for series parallel graphs that is presented in [8] (Section 4.1): rules 1a and 2 – 18 form the set of rules for series parallel graphs. Rule 1 consists of two parts, which distinguish between the case in which none of the edges that are involved in the reduction have label B (rule 1a), and the case in which at least one of the edges that are involved in the reduction has label B (rule 1b).

Rules 21 – 23 are necessary to reduce long sequences of 'small' biconnected components as shown in Figure 2 quickly enough: in such a sequence, only two concurrent reductions are possible without rules 21 – 23.

Rule 20 is necessary for reducing dangling edges, i.e. edges of which one end vertex has degree one: these edges may not appear in series parallel graphs, unless if they can be reduced with rule 1 or the graph consists of one edge. However, in graphs of treewidth at most two, these edges can exist.

In rule 20, we pose a degree constraint of eight on the terminal vertex. This means that if we rewrite a terminal subgraph G_1 in G which is isomorphic to the left-hand side of rule 20, then the terminal vertex of G_1 has degree at most eight in G. This degree constraint is added to avoid problems with writing conflicts in the parallel algorithm. It also makes the presence of rule 19 necessary: without rule 19, a large star-like graph can not be reduced.

Note that, in rules 3 – 18, we pose degree constraints on the edges between terminals (see Section 4.1 of [8]).

The following lemma was first proved by Duffin [12], although not precisely in this form.

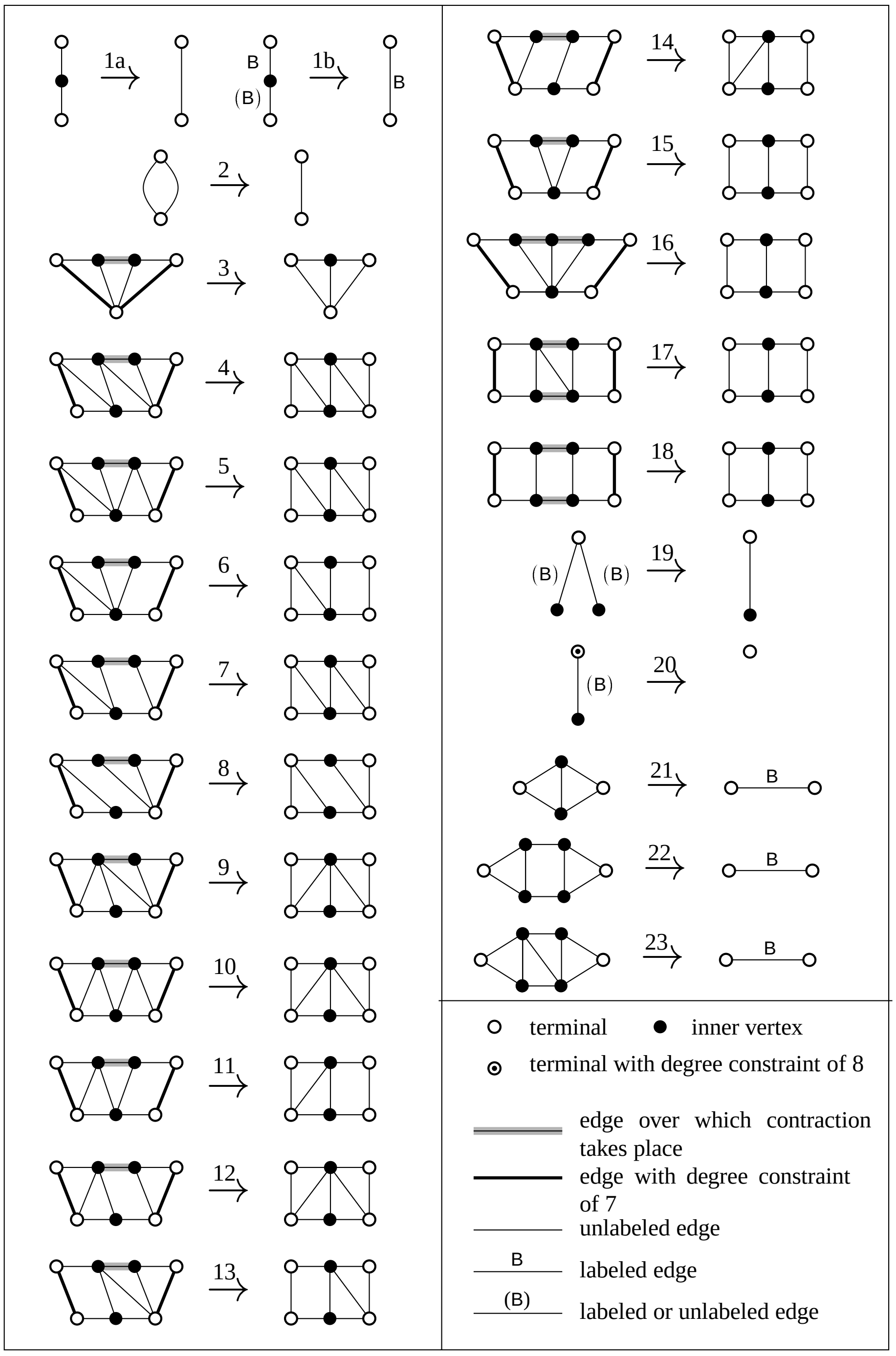
Lemma 3.1. *A graph is series parallel if and only if it can be reduced to a single edge by applying any sequence of reductions by rule 1a and 2. A source-sink labeled graph is series parallel if and only if it can be reduced to a single edge by applying any sequence of reductions by rule 1a and 2, such that neither s or t is the inner vertex in rule 1a in any of the reductions.*

This shows safeness of rules 1a and 2 for series parallel graphs. In [8], it was shown that rules 3 – 18 are safe for series parallel graphs.

Hence, given a connected B-labeled graphh G, a match to a reduction rule $r = (H_1, H_2) \in \mathcal{R}_{\text{tw}}$ 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 $G = G_1 \oplus G_2$,
- 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, and
- if r is rule 20, then the terminal vertex u in G_1 has degree at most eight in G.

Lemma 3.2. *The set $\mathcal{R}_{\text{tw}}$ of reduction rules is safe for treewidth at most two on connected* B*-labeled graphs.*



1a
B
(B)
1b
B
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
(B)
(B)
19
(B)
20
21
B
22
B
23
B
terminal
inner vertex
terminal with degree constraint of 8
edge over which contraction takes place
edge with degree constraint of 7
unlabeled edge
B
labeled edge
(B)
labeled or unlabeled edge

Figure 1: Reduction rules for treewidth at most two on connected, B-labeled graphs

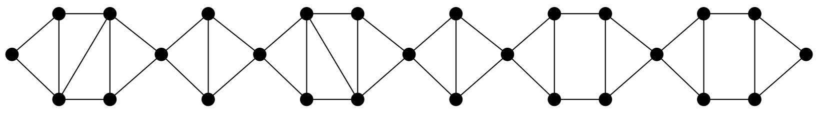


Figure 2: A chain of ‘small’ biconnected components

Proof. Let G be a connected B-labeled graph, let $r \in \mathcal{R}_{\text{tw}}$, and suppose G contains a match H to r. Let G' be the graph obtained from G by applying the reduction corresponding to match H. We show that G has treewidth at most two if and only if G' has treewidth at most two. Note that a B-labeled graph has treewidth at most two if and only if all its blocks have treewidth at most two.

First suppose r is one of the rules 2 – 18. Then H is contained in one of the blocks of G. Let B denote this block (note that B is a non-trivial block and H is also a match in B), and let B' be the graph obtained from B by applying the rule. Then B' is a block of G'. Therefore, it suffices to show that B has treewidth at most two if and only if B' has treewidth at most two. This follows from Lemma 2.2 and the fact that rules 2 – 18 are safe for series parallel graphs.

Suppose r is rule 1. If all vertices of H are contained in one block B, then this is a non-trivial block. It easily follows that G has treewidth at most two if and only if G' has treewidth at most two: rule 1a is safe for series parallel graphs, and if an edge in H has a B-label then neither G nor G' have treewidth at most two.

Suppose the vertices of H are not in one block. Then the two edges of H are separate blocks, and they are both bridges (hence they both have treewidth at most two). This implies that the new edge is a block in G', and it is also a bridge in G' (hence it also has treewidth at most two). This shows that G has treewidth at most two if and only if G' has treewidth at most two.

It is easy to see that rules 19 and 20 are safe for TW2: if r is rule 19 or 20, then the blocks of G have treewidth at most two if and only if the blocks of G' have treewidth at most two.

Suppose r is one of the rules 21, 22 and 23. Let x and y be the terminals of H. Suppose G has treewidth at most two. If G contains a path between the terminals of H which avoids the inner vertices of H, then G contains a K_4 minor, hence this is not the case. This means that x and y are cut vertices of G, and hence H is a block of G. This implies that in G', the edge between x and y is a bridge of G', and hence it is a block of G' which has treewidth at most two. Hence G' has treewidth at most two.

If G' has treewidth at most two, then the edge between x and y is a bridge, and hence is a block with treewidth at most two. This implies that H is a block in G. As H has treewidth at most two, we have that G has treewidth at most two. $\square$

3.2 A Lower Bound on the Number of Matches

In this section, we show that each connected graph G of treewidth at most two with at least one edge has at least $\Omega(|E(G)|)$ matches.

We first prove the following lemma.

Lemma 3.3. *Let G be a connected* B*-labeled graph and let $v \in V(G)$ such that v has degree at* *most eight. Then the number of matches to rules 1 – 23 in G which contain v is at most some integer constant k.*

Proof. We give a very rude bound which is probably far too large, but easy to prove. Note that all inner vertices of left-hand sides of rules 1 – 23 have degree at most eight. Let G_1 be a match in G which contains v. It can be seen that all vertices and edges in G_1 are reachable by a path P from v to this vertex or edge, such that all vertices on the path except possibly the first and the last one are inner vertices of G_1, or are terminals of G_1 with degree at most eight. Hence each vertex on such a path, except the last one, has degree at most eight. Furthermore, the path has length at most seven, as each left-hand side of a reduction rule has at most eight vertices. Therefore, the number of vertices and edges in G which are reachable from v by such a path is at most 8^7. This implies that there is at most a constant number of matches containing v. □

Let $G = (V,E)$ be a connected B-labeled graph, suppose the treewidth of G is at most two, and let $|E| \geq 1$.

A *dangling edge* in G is an edge $e = \{u,v\}$ for which either u or v has degree one. If u has degree one, then e is called a dangling edge of v. A *star* is a graph consisting of one vertex with dangling edges. A *pseudo block* is a graph which is a star, or consists of one block with dangling edges, i.e. G consists of a block of which some vertices have dangling edges.

We divide G into pseudo blocks as follows. If G is a star, then G itself is the only pseudo block. Otherwise, let $\mathcal{B}$ denote the set of all blocks of G, and let $\mathcal{B}' \subseteq \mathcal{B}$ be the set of all blocks which are non-trivial or have two or more cut vertices. Note that $\mathcal{B}'$ contains exactly all blocks which are not dangling edges, and each dangling edge has an end point in one of the blocks in $\mathcal{B}'$. Assign each dangling edge to a block in $\mathcal{B}'$ which contains one of its end points. A pseudo block of G consists of a block in $\mathcal{B}'$ with the dangling edges assigned to it. Let $\mathcal{PB}$ be the set of all the pseudo blocks. For each pseudo block $PB \in \mathcal{PB}$, we call the block of PB which is in $\mathcal{B}'$ the *underlying* block of PB.

The vertices that are contained in two or more pseudo blocks are called the *strong cut vertices* of G, and we denote the set of all strong cut vertices by $\mathcal{S}$. If v is a strong cut vertex, then v is a cut vertex of G and v is contained in the underlying block of each pseudo block it is contained in.

Let M denote the set of all matches in G.

Lemma 3.4. *If G consists of one pseudo block, then $|E| \leq k_0|M|$ for some integer constant k_0.*

Proof. Let $m = |E|$. If $m = 1$, then G contains a match to rule 20. Suppose $m \geq 2$. If G is a star, then G has $m(m-1)/2$ matches to rule 19, and hence $m \leq 2|M|$. If G consists of an edge with one or more dangling edges at each end point, then either $m = 3$ and G has a match to rule 1 or 19, or $m > 3$ and G has at least $(m-1)(m-3)/8$ matches to rule 19 (at least $(m-1)/2$ edges are dangling edges of the same end point). Hence $m \leq 9|M|$.

Suppose G consists of a non-trivial block B with dangling edges D. Note that B has no edges labeled B. Let D_1 denote the dangling edges which are dangling edges of some vertex of B that has one dangling edge, and let $D_{\geq 2}$ denote the other dangling edges. Note that G has at least $|D_{\geq 2}|/2$ matches to rule 19, and hence $|D_{\geq 2}| \leq 2|M|$.

Consider block B. As B is series parallel and has at least two edges, it contains at least $|E(B)|/139$ matches to rules 1 – 18 (Lemma 4.4 in [8]). Consider the set M_{sp} of all these matches. Let $H \in M_{\text{sp}}$. Either H is a match in G or not. If H is not a match in G, we call H a disturbed match.

If H is disturbed, then either an inner vertex v of H has one or more dangling edges, or a terminal vertex v of H which has degree at most seven in B has one or more dangling edges. In both cases, v has degree at most seven in B. Furthermore, if v has one dangling edge, then it has degree at most eight in G and hence e is a match to rule 20. If v has two or more dangling edges, then two of these edges form a match to rule 19 in G. By Lemma 3.3, the number of matches in G which contains v is at most k. Hence the number of disturbed matches is at most k times the number of matches to rules 19 and 20 in G. This means that we can derive the following upper bound for $|M_{\text{sp}}|$.

$$\begin{aligned}|M_{\text{sp}}| &= |\{\text{non-disturbed matches}\}| + |\{\text{disturbed matches}\}| \\ &\le |\{\text{matches to rules } 1-18\}| + k \cdot |\{\text{matches to rules 19 and 20}\}| \\ &\le k|M|\end{aligned}$$

Furthermore, $|D_{\ge 2}| \le 2|M|$ and $|D_1| \le |V(B)| \le |E(B)|$. Hence $m = |E(B)| + |D_1| + |D_{\ge 2}| \le 2|E(B)| + 2|M| \le 278|M_{\text{sp}}| + 2|M| \le 278(k+1)|M|$, so the lemma holds with $k_0 = 278(k+1)$. □

In the following discussion, we denote for each pseudo block PB the set of matches in PB by M_{PB}. A match in M_{PB} is either a match in G, in which case it is called a *non-disturbed* match, or it is not a match in G, in which case it is called a *disturbed* match. The set of non-disturbed matches in M_{PB} is denoted by M_{PB}^{nd}, and the set of disturbed matches in M_{PB} is denoted by M_{PB}^{d}. Note that $M_{PB}^{\text{nd}} \subseteq M$. The union over all pseudo blocks of M_{PB}^{nd} (M_{PB}^{d}) is denoted by M_G^{nd} (M_G^{d}).

The only matches in G which involve two or more pseudo blocks may be matches to rules 1 or 19: both can involve only two pseudo blocks. Let M_2 denote the set of matches in G which involve two pseudo blocks. Note that $M = M_G^{\text{nd}} \cup M_2$.

Lemma 3.4 implies the following result.

Corollary 3.1. *For each pseudo block PB,* $|E(PB)| \le k_0|M_{PB}|$.

This implies that $|E| \le k_0|M| = k_0|M_G^{\text{nd}}| + k_0|M_G^{\text{d}}|$. If we prove that $|M_G^{\text{d}}| \le k|M|$ for some k, then the proof of the main result of this section is finished. We prove this by accounting each disturbed match to either a non-disturbed match or a match in M_2, such that each match in M has at most a constant number of disturbed matches accounted to it.

Consider a disturbed match H in M_{PB}^{d}. Then there is a strong cut vertex v in PB for which either

- v is an inner vertex of H,
- H is a match to rule 20 and v is a terminal of H, v has degree at most eight in PB and v has degree more than eight in G, or

- H is a match to one of the rules 3 – 18, v is a terminal of H, v has degree at most seven in PB, and v has degree more than seven in G.

If one of these cases holds for a strong cut vertex v and a disturbed match H, we say that *v disturbs H*.

Lemma 3.5. *Each strong cut vertex disturbs at most k matches in each pseudo block it is contained in.*

Proof. Let PB be a pseudo block and let v be a strong cut vertex in PB. Let H be a match that is disturbed by v. Note that v has degree at most eight in PB. Hence, by Lemma 3.3, there are at most k matches in PB which contain v. This means that v disturbs at most k matches. □

We divide the pseudo blocks of G into different classes, which correspond to the type of pseudo block that they are contained in. After that, we give for each class an upper bound on the number of edges in this class with respect to the number of matches in G. Therefore, we first construct a *pseudo block tree* $T = (N,F)$ as follows.

$$N = \mathcal{PB} \cup S$$
$$F = \{\{v,PB\} \mid v \in S \wedge PB \in \mathcal{PB} \wedge v \in V(PB)\}$$

Hence T contains as its vertices the pseudo blocks and strong cut vertices of G, and there is an edge between two vertices in T if and only if one of them is a cut vertex v, the other one is a pseudo block PB, and v is contained in PB. Note that the degree of a strong cut vertex in T equals the number of pseudo blocks it is contained in, and the degree of a pseudo block in T equals the number of strong cut vertices it contains. We call a pseudo block a *degree d pseudo block* if its corresponding node in N has degree d in T. A degree one pseudo block is also called a leaf pseudo block. Note that each leaf pseudo block has at least two edges (if it had only one edge, then it would be a dangling edge of one of the blocks it shares a vertex with).

We partition the set $\mathcal{PB}$ of pseudo blocks into four sets: $\mathcal{PB}_0$, $\mathcal{PB}_1$, $\mathcal{PB}_2$ and $\mathcal{PB}_{\geq 3}$. For $i = 0,1,2$, $\mathcal{PB}_i$ is the set of degree i pseudo blocks. The set $\mathcal{PB}_{\geq 3}$ is the set of all degree d pseudo blocks with $d \geq 3$.

Degree zero pseudo blocks. A degree zero pseudo block has no disturbed matches, as it has no strong cut vertices.

Degree one pseudo blocks. As a degree one pseudo block contains only one strong cut vertex, it has at most k disturbed matches. It suffices to show that each such block contains at least one non-disturbed match: each disturbed match can be accounted to a non-disturbed match such that at most k disturbed matches are accounted to each non-disturbed match in a degree one pseudo block.

Lemma 3.6. *Each degree one pseudo block contains at least one non-disturbed match.*

Proof. Let *PB* be a leaf pseudo block, let x denote the strong cut vertex in *PB*. We show that $|M^{\text{nd}}_{PB}| \geq 1$. If the underlying block of *PB* is an edge e, then the end point of e which is not x has at least one dangling edge, and hence there is at least one match to rule 19 or 20. This match is not disturbed by x, so $|M^{\text{nd}}_{PB}| \geq 1$.

Suppose the underlying block B of *PB* is non-trivial. Note that $x \in V(B)$. Let y be a neighbor of x in B. By Lemma 9 from [13], (B,x,y) is series parallel and hence it has at least one match to rules 1 or 2 which does not have x or y as inner vertex. This means that *PB* has at least one match to rule 1, 2, 19 or 20 which does not have x as an inner vertex, and hence is not disturbed in G. Hence $|M^{\text{nd}}_{PB}| \geq 1$. □

Degree two pseudo blocks. Consider the set PB_2. Note that each degree two pseudo block contains at most $2k$ disturbed matches.

We split PB_2 in two parts PB_2^{nt} and PB_2^{t}. The first set contains all degree two pseudo blocks of which the underlying block is non-trivial, and the second set contains all other degree two pseudo blocks (i.e. the degree two pseudo blocks of which the underlying block is trivial).

First consider the degree two pseudo blocks in PB_2^{nt}. It suffices to show that each such pseudo block contains at least one non-disturbed match: then we can account each disturbed match to such a non-disturbed match.

Lemma 3.7. *Each degree two pseudo block* $PB \in PB_2^{\text{nt}}$ *contains at least one non-disturbed match.*

Proof. The idea of the proof is the following: if a degree two pseudo block contains no non-disturbed matches to one of the rules 1 — 20, then it must contain a match to one of the rules 21 – 23. The complete proof consists of a long and tedious case analysis, which we omit here (see [11] for the complete proof). □

Next consider the pseudo blocks in PB_2^{t}. Let $PB \in PB_2^{\text{t}}$. If *PB* contains a non-disturbed match, then the disturbed matches can be accounted to this match and we are done.

Suppose *PB* contains no non-disturbed match. Then *PB* consists of one edge $e = \{u,v\}$ which is the underlying block, with at most one dangling edge at both u and v. Note that both u and v are strong cut vertices with degree at least two.

If u or v is a strong cut vertex with degree three or more, then we can use Lemmas 2.3 and 3.6 and account all disturbed matches in *PB* to a non-disturbed match in a leaf pseudo block in such a way that each such non-disturbed match has at most $6k$ disturbed matches of pseudo blocks like *PB* accounted to it.

Suppose u and v are strong cut vertices with degree two. Suppose u is contained in another pseudo block PB'. If PB' is a degree two pseudo block of the same type as *PB* (i.e. PB' is a degree two pseudo block of which both strong cut vertices have degree two), then *PB* and PB' together contain a match to one of the rules 1, 19 or 20, and we can account the disturbed matches in *PB* (and PB') to this match. This match has at most $4k$ of disturbed matches of pseudo blocks like *PB* accounted to it.

If PB' is not of the same type, then a detailed case analysis, based on the type of pseudo block PB' is, shows that the disturbed matches of *PB* can be accounted to a non-disturbed match in another pseudo block, such that each such non-disturbed match has at most a constant number of disturbed matches accounted to it. We do not give the full details here; they can be found in [11].

Degree ≥ 3 pseudo blocks. Let $d \geq 3$ and let PB be a degree d pseudo block. Note that the underlying block B of PB is non-trivial. By Lemma 3.5, there are at most $d \cdot k$ disturbed matches in PB. Lemma 2.3 shows that the sum over all degrees of the pseudo blocks in $PB_{\geq 3}$ is at most three times the number of leaf pseudo blocks. By Lemma 3.6, each leaf pseudo block contains at least one match in M. Hence we can account each disturbed match in a degree ≥ 3 pseudo block to a non-disturbed match in a leaf pseudo block such that each such non-disturbed match has at most $3k$ disturbed matches in degree ≥ 3 pseudo blocks accounted to it.

We have now shown that each disturbed match in M^{d}_{PB} can be accounted to either a non-disturbed match in M^{nd}_{PB} or a match in M_2, such that each match in $M = M^{\mathrm{nd}}_{PB} \cup M_2$ has at most a constant number of disturbed matches accounted to it. This proves the following result.

Lemma 3.8. *There is a constant $c > 0$, such that each connected* B*-labeled graph $G = (V,E)$ with treewidth at most two and $|E| \geq 1$ contains at least $c|E|$ matches.*

3.3 A Lower Bound on the Number of Enabled Matches

In this section we show that, in each connected B-labeled graph G of treewidth at most two with at least one edge, a number of at least $c|E(G)|$ matches to rules in $\mathcal{R}_{\mathrm{tw}}$ can be found in $O(1)$ time with $O(|E(G)|)$ processors. This is done in the same way as for series parallel graphs in Section 4.3 of [8]; only rule 19 gives extra complications.

The finding of the rules is done as follows. Given an edge e it can easily be checked whether e can occur in an application of one of the rules 1, 3 – 18 or 20 – 23: 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). For rules 2 and 19, not all matches are found. 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 (see also [8]). For rule 19, every edge $e = \{u,v\}$ of which end point u has degree one searches in the adjacency list of v for all edges which have distance at most ten to e in this list. The edge e proposes an application of rule 19 if it finds an edge $e' = \{v,w\}$ for which w has degree one. (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. The proof is similar to, but more complicated than the proof for series parallel graphs.

Let G be a B-labeled graph $G = (V,E)$ given by some adjacency list representation. Recall from [8] that an edge e is bad if it has a parallel edge, but all its parallel edges have distance at least 21 in the adjacency lists of the end points of e.

A dangling edge e is called a *bad dangling edge* if it is incident with a vertex v that has two or more dangling edges, but in the adjacency list of v, these dangling edges have distance at least 21 to e. Note that an edge is bad if and only if it occurs in a match to rule 2, but not in an enabled match to rule 2. A dangling edge is bad if and only if it occurs in a match to rule 19, but not in a enabled match to rule 19.

Lemma 3.9. *Let $G = (V,E)$ be a* B*-labeled graph of treewidth at most two, given by some adjacency list representation. The graph G has at most $|E|/5$ bad edges and at most $|E|/10$ bad dangling edges.*

Proof. For the bound on the bad edges, see [8], Lemma 4.6. Consider the bad dangling edges. Let $v \in V(G)$. If the adjacency list of v has length at most 20, then v does not have any bad dangling edges. If the adjacency list of v has length more than 20, then each 20 successive entries in the (cyclic) adjacency list contain at most one bad dangling edge. Hence there are at most $\deg(v)/20$ bad dangling edges in the adjacency list of v. If we sum over all vertices, we get that the number of bad dangling edges is at most $|E|/10$. □

Lemma 3.10. *There is a constant $c' > 0$ for which each connected* B*-labeled graph G of treewidth at most two with at least one edge has at least $c'|E(G)|$ enabled matches.*

Proof. We use the same idea as in the proof of Lemma 4.7 of [8]. Let $G = (V,E)$ be a B-labeled graph with at least one edge. Let $n = |V|$ and let $m = |E|$. Let de denote the number of dangling edges of G of which one end point has at least two dangling edges, i.e. the dangling edges which occur in a match to rule 19. We distinguish between three cases:

1. $m \geq 4n$,
2. $de \geq m/5$, and
3. $m < 4n$ and $de < m/5$.

Case 1. Suppose $m \geq 4n$. As G has treewidth at most two, the underlying simple graph has at most $2n$ edges. This means that at least $m \Leftrightarrow 2n$ edges are parallel to another edge. At most $m/5$ of these are bad edges, hence at least $m \Leftrightarrow 2n \Leftrightarrow m/5 \geq 4m/5 \Leftrightarrow m/2 = 3m/10$ edges occur in an enabled match to rule 2. This means that there are at least $3m/20$ enabled matches to rule 2 in G.

Case 2. Suppose that $de \geq m/5$. Of the de dangling edges which occur in a match to rule 19, at most $m/10$ are bad. Hence at least $de \Leftrightarrow m/10 \geq m/10$ of these dangling edges occur in a enabled match to rule 19. This means that there are at least $m/20$ enabled matches to rule 19 in G.

Case 3. Suppose that $m < 4n$ and $de < m/5$. If G is a star, then G contains at least $m/20$ enabled matches to rule 19.

Suppose G is not a star. For each $v \in V(G)$, remove all dangling edges adjacent to v except one (if there is at least one). Furthermore, for each pair u,v of vertices in G which have two or more parallel edges between them, do the following. If u and v both have two or more neighbors, then remove all edges except one between u and v. If u or v has only one neighbor, then remove all but two edges between u and v. Let G' denote the resulting graph, and let $n' = |V(G')|$, $m' = |E(G')|$.

Note that if G' contains a match to rule 2, then one of the terminals of this match has degree two, and hence this match is enabled. Furthermore G' has no matches to rule 19. Hence all matches are enabled.

We express n' and m' in terms of m: $n' \geq n \Leftrightarrow de > m/4 \Leftrightarrow m/5 = m/20$. As G is not a star, it follows that $n' \geq 2$. Furthermore, G' is connected, and hence $m' \geq n' \Leftrightarrow 1 \geq n'/2 \geq m/40$.

Note that $m' \geq 1$ (since G is not a star). By Lemma 3.8, G' contains at least $c \cdot m' \geq c/40 \cdot m$ matches. Let M denote the set of all matches in G'. Each of these matches is either an enabled match in G, or it is not a enabled match in G. We call the first set the set of non-disturbed matches, denoted by M_{nd}, and the second the set of disturbed matches, denoted by M_{d}. Let M_{new} denote the set of enabled matches in G which are not in G'. Note that the set of enabled matches in G is $M_{\mathrm{nd}} + M_{\mathrm{new}}$.

We account each match in M_{d} to a match in $M_{\mathrm{nd}} + M_{\mathrm{new}}$, such that each match has at most a constant number of disturbed matches accounted to it. By the fact that $|M_{\mathrm{nd}}| + |M_{\mathrm{d}}| \geq c/40 \cdot m$, this proves the lemma.

Consider a match $H \in M_{\mathrm{d}}$. If H is a match to rule 2, then the terminal v of H which has degree two in G' has degree more than two in G. If H is a match to one of the other rules, then either H contains an inner vertex v which has dangling edges in G or which is incident with parallel edges in G, or v is a terminal which has degree $d \leq 8$ in G', but has degree more than d in G. In all cases, there is a vertex $v \in V(H)$ which has degree $d \leq 8$ in G' and has degree more than d in G.

Since v has larger degree in G than in G', it must be the case that in G, v has two or more dangling edges, or v is incident with parallel edges. If the adjacency list of v has length at most 20, then there are two edges incident with v which form a enabled match to rule 2 or rule 19. Let H_v denote this match, note that $H_v \in M_{\mathrm{new}} \cup M_{\mathrm{nd}}$. If the adjacency list of v has length more than 20, then consider a sublist of length 20 of this list. If this sublist contains two or more dangling edges, then two of these form a enabled match to rule 19 in G, and hence this match is in M_{new}. Let H_v denote this match. If the sublist contains at most one dangling edge, then 20 or 21 of the places in this sublist contain an edge e between v and a neighbor of v. As v has at most eight distinct neighbors, there must be at least two edges with the same end points in the sublist. Two of these edges correspond to a enabled match to rule 2 in G. Let H_v again denote this match, and note that $H_v \in M_{\mathrm{new}} \cup M_{\mathrm{nd}}$.

Note that, as v has degree at most eight in G', it is contained in at most k matches in M and hence in M_{d} (Lemma 3.3). For each match H in M_{d}, account H to a match H_v of a vertex $v \in V(H)$ which has degree $d \leq 8$ in G' and degree more than d in G. In this way, each match to rule 19 in $M_{\mathrm{new}} \cup M_{\mathrm{nd}}$ has at most k matches accounted to it, and each match to rule 2 in $M_{\mathrm{new}} \cup M_{\mathrm{nd}}$ has at most $2k$ matches accounted to it (at most k for each end point). This completes the proof. $\square$

3.4 Constructing a Tree Decomposition

In this section we show how, in the second phase of the algorithm, a tree decomposition of the current graph is maintained in each construction round. The tree decomposition that is maintained is of a special form, in order to make the constructions easier.

Definition 3.1 (Special Tree Decomposition). *Let $G = (V,E)$ be a connected B-labeled graph with treewidth at most two. Let $TD = (T,X)$ be a tree decomposition of width two of G with $T = (I,F)$ and $X = \{X_i \mid i \in I\}$. Then TD is a* special tree decomposition *of G if it satisfies the following conditions.*

1. *For each vertex $u \in V$ there is a unique node i with $X_i = \{u\}$, called the node associated with u.*
2. *Each edge $e \in E$ with end points u and v has a node i with $X_i = \{u,v\}$ associated with it. Distinct edges have distinct associated nodes.*
3. *Let u be a cut vertex of G, let i denote the node associated with u. Then each component of $T[I \Leftrightarrow \{i\}]$ contains vertices of at most one component of $G[V \Leftrightarrow \{u\}]$.*
4. *Let e be a bridge of G with end points u and v and let i be the node associated with e. Then each component of $T[I \Leftrightarrow \{i\}]$ contains vertices of exactly one component of $(V, E \Leftrightarrow \{e\})$.*
5. *Let $u,v \in V$. If there is an edge between u and v, and $\{u,v\}$ is a minimal x,y-separator for some vertices x and y, then there is a node i associated with some edge between u and v such that x and y occur in different components of $T[I \Leftrightarrow \{i\}]$. ($\{u,v\}$ is a minimal x,y-separator if x and y are in different components of $G[V \Leftrightarrow \{u,v\}]$, but in the same component of $G[V \Leftrightarrow \{u\}]$ and of $G[V \Leftrightarrow \{v\}]$.)*
6. *For each two adjacent nodes $i, j \in I$, $\big||X_i| \Leftrightarrow |X_j|\big| = 1$, unless if $X_i = X_j = \{u,v\}$ and i and j are nodes associated with different edges between u and v.*
7. *For each $u,v \in V$, the nodes associated with edges between u and v induce a subtree of T.*

We use the following data structure for storing a special tree decomposition during the second phase of the algorithm. We store a list containing all nodes of the tree decomposition. Each node i has an adjacency list which contains an entry for each neighbor of i. An entry for neighbor j in the adjacency list of i contains a pointer to j, the contents X_j of node j, and a pointer to the entry of i in the adjacency list of j. Furthermore, for each vertex and edge in the graph, we keep a pointer to the node associated with it.

Phase two of the algorithm starts with a graph G with one vertex v. It simply constructs a tree decomposition of one node which contains v. Note that this tree decomposition satisfies conditions 1 – 7, and hence is a special tree decomposition.

Suppose at some construction round in the algorithm, we are given a B-labeled graph G and a special tree decompositon $TD = (T,X)$ of G. Suppose some processor has to undo rule $r = (H_1,H_2) \in R_{\text{tw}}$, and has a match $G_2 = (V_2,E_2,X)$ to H_2 which is replaced by a match $G_1 = (V_1,E_1,X)$ to H_1. Let $T = (I,F)$ and $X = \{X_i \mid i \in I\}$. The construction algorithm that is executed to locally modify the tree decomposition consists of 23 steps which are executed consecutively. Each round corresponds to a rule in R_{tw}: if the processor has to undo rule number r, then it is only active in step r. We describe the algorithm per step.

Step 1. If r is not rule 1, then the algorithm is idle in this step. Suppose r is rule 1, and suppose $V_2 = \{a,b\}$, $E_2 = \{e\}$ and $V_1 = \{a,b,c\}$. See part I of Figure 3 (labelings of edges are not shown). Let i and j be the nodes of TD associated with a and b, and let k be the node associated with e.

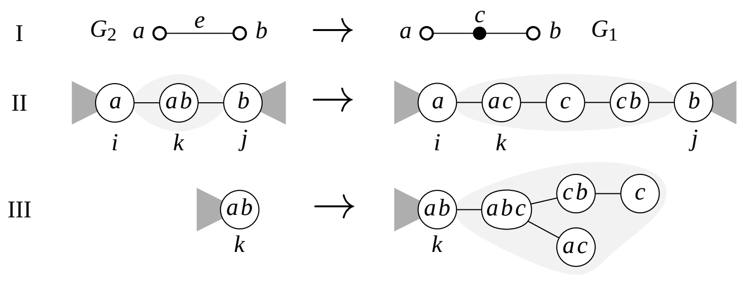


Figure 3: The construction for rule 1.

If e is a bridge in G, then a and b are cut vertices. The node k associated with e separates TD in different components corresponding to the components of $G' = (V, E \Leftrightarrow \{e\})$. Note that G' has exactly two components, one containing a and one containing b. ¿From the conditions of Defiition 4 we can derive that TD contains a subtree as shown in the left-hand side of part II of Figure 3. We replace this by the subtree shown in the right-hand side of part II (the light-gray parts of the tree decompositions are the parts that are involved in the modification). Note that the new edges are bridges and c is a cut vertex. Hence the new tree decomposition is a special tree decomposition.

Consider the case that e is not a bridge. Then node k does not necessarily have i and j as its only neighbors. If this is indeed not the case, we add an extra node l as new neighbor of k, with $X_l = \{a,b,c\}$, and we add some other nodes to fulfil conditions 1 – 7. See part III of Figure 3. Note that the new tree decomposition is indeed a special tree decomposition (c is not a cut vertex, the new edges are not bridges, and the sets $\{a,c\}$ and $\{c,b\}$ are not minimal x,y-separators).

Hence the following is done. If node k has as its only neighbors node i and k, the constructiton of part II of Figure 3 is applied. The new nodes are added, the contents of X_k is changed, and in the adjacency lists of i and j, the entries for node k are modified. Furthermore, the nodes with contents $\{a,c\}$, $\{b,c\}$ and $\{b\}$ are the nodes associated with the edge between a and c, the edge between b and c, and vertex c, respectively.

If k does not have only i and j as its neighbors, the algorithm applies the construction of part III of Figure 3. This can be done by adding the new nodes with their adjacency lists, and adding an entry for the new node adjacent to k at the end of the adjacency list of k. The new nodes are the nodes associated with the new edges and vertices.

It can be seen that this construction is correct, and that it takes $O(1)$ time on one processor. Furthermore, the construction does not interfere with other constructions for rule 1 which take place simultaneously: edge e is not involved in any other reduction at the same time, and hence node k is not involved in any other reductions that are performed in step 1. Nodes i and j may be involved in other applications of rule 1, however, only the contents of entries in the adjacency list of i and j are modified, and this can be done in different places of the adjacency list at the same time without concurrent reading or writing.

Step 2. If r is not rule 2, then the algorithm is idle in this step. Suppose r is rule 2. The construction is very simple. Part I of Figure 4 shows G_1 and G_2. Let i be the node associated with e. Then we can apply the construction of part II of Figure 4. The newly added node is the node associated with edge e'. Note that condition 7 is satisfied, and hence TD is a special tree decomposition. It is easy to see that this construction can be done in $O(1)$ time and that it does not interfere with other constructions in step 2.

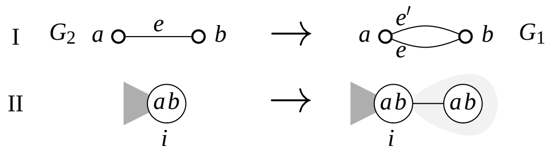


Figure 4: The construction for rule 2.

Step 3. Suppose r is rule 3 (otherwise, the algorithm is idle in this step). Let G_1 and G_2 be as depicted in part I of Figure 5. Let i, j and k be the nodes associated to edge e_1, e_2 and e_3, respectively. By a detailed analysis, using the properties described in Definition 3.1, it can be shown that T contains a subtree as depicted in the left-hand side of part II of Figure 5. The possible adjacencies of the nodes associated with a, b, c, and d are denoted by dashed lines.

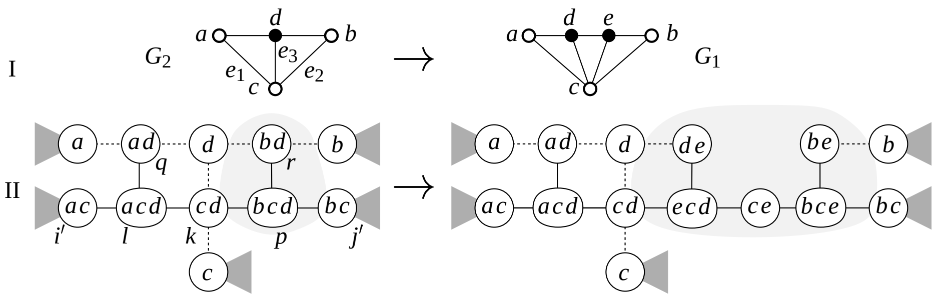


Figure 5: The construction for rule 3. Dashed lines denote possible adjacencies.

We replace a part of this subtree by a new subtree, as is depicted in the right-hand side of part II of Figure 5. The dashed lines again denote possible adjacencies, which are the same as in the left-hand side. Note that the new tree decomposition satisfies conditions 1 – 7 of Definition 3.1, and hence it is a special tree decomposition. It is easy to see that this construction can be done in $O(1)$ time, and that the construction does not interfere with other constructions for rule 3. (Note that there are no write conflicts, as each edge that is involved in this reduction is not involved in another simultaneous reduction.)

Steps 4 – 18. The constructions for rules 4 – 18 are similar to the construction for rule 3, so we do not describe them. The rules of which the right-hand side contains a chordless four-cycle are a bit different, as there are two possibilities for the structure of the tree decomposition. As an example, we depict the two possible constructions for rule 6 in Figure 6.

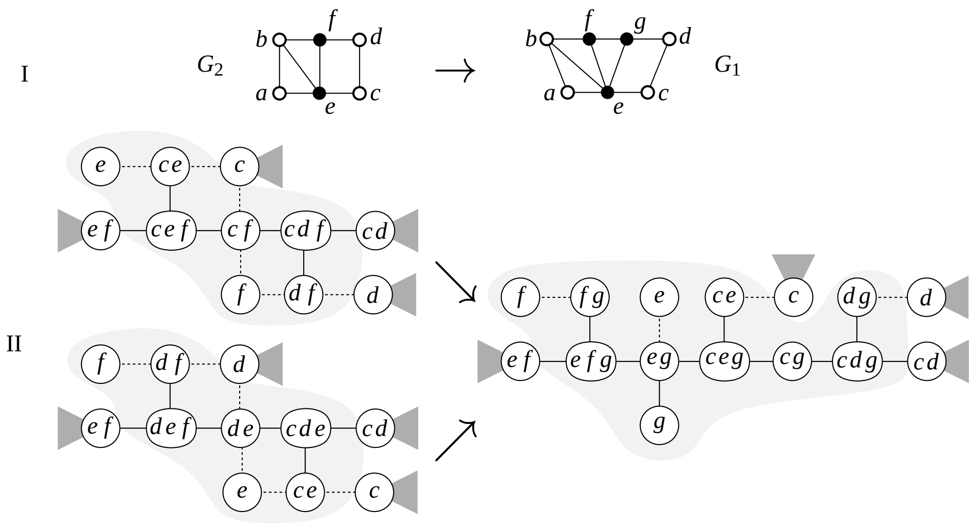


Figure 6: The construction for rule 6. Dashed lines denote possible adjacencies.

Step 19. Suppose r is rule 19. Let G_1 and G_2 be as depicted in part I of Figure 7 (labelings of edges are not shown). Note that edge e is a bridge, and hence the tree decomposition contains a subtree as depicted in the left-hand side of part II of Figure 7 (see also step 1). We replace this subtree by the subtree depicted in the right-hand side of part II of Figure 7. Note the resulting tree decomposition is special.

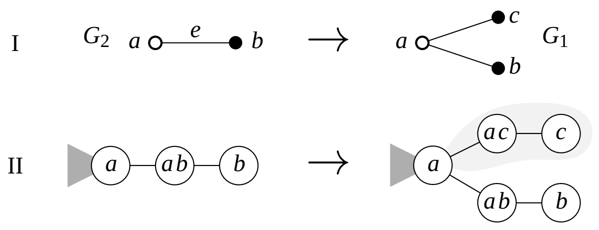


Figure 7: The construction for rule 19.

In order to make the construction non-interfering, it is done as follows. We make two new nodes with contents $\{a, c\}$ and $\{c\}$, respectively, which are adjacent to each other. Furthermore, we make a new entry in the adjacency list of the node associated with a. The new entry is for the node associated with edge $\{a, c\}$. It is added between the entry for the node associated with edge e and its right neighbor in the list. In this way, no two constructions for rule 19 try to modify the same entry of the adjacency list of node i.

Step 20. Suppose r is rule 20. Let G_1 and G_2 be as depicted in part I of Figure 8. Let i denote the node associated with a. We apply the construction depicted in part II of Figure 8. Note that the resulting tree decomposition is special.

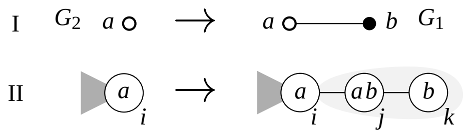


Figure 8: The construction for rule 20.

In order to make the algorithm non-interfering in step 20, the construction is as follows. Let H be a B-labeled terminal graph such that $G = G_2 \oplus H$. Note that, in $G_1 \oplus H$, vertex a has degree at most eight. We divide step 20 into eight substeps, which are executed subsequently. First, edge e computes its rank k in the adjacency list of a in $G_1 \oplus H$. Then, in substep k, the construction is applied by adding new nodes j and l with $X_j = \{a,b\}$ and $X_l = \{a\}$ which are adjacent to each other, and making j adjacent to i: an extra entry for node j is added at the end of the adjacency list of node i. In each substep, at most one such construction that involves vertex a is applied, and hence the algorithm is non-interfering and runs in $O(1)$ time.

Steps 21 – 23. Rules 21, 22 and 23 are very similar to each other. We depict the construction only for rule 21 in Figure 9. It is easy to see that the new tree decomposition is special, and that the construction can be done in $O(1)$ time and is non-interfering.

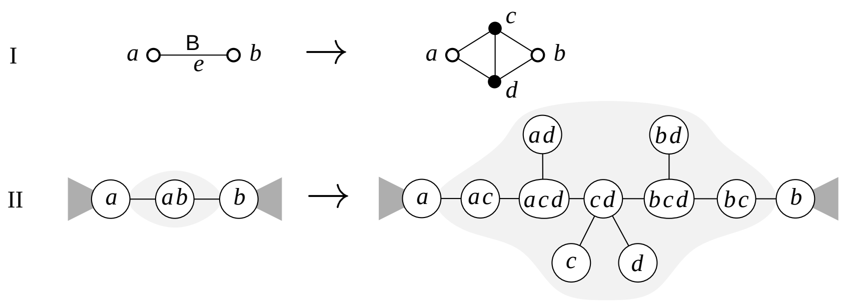


Figure 9: The construction for rule 21.

4 Additional Results

We can use the algorithm of Section 3 for the same problem, but without requiring that the input graph is connected. To this end, we use a technique of [9]: from each connected component of the graph we select one vertex. Then we add a new dummy vertex to the graph, and make all selected vertices adjacent to this dummy vertex. The new graph is connected, and has treewidth at most two if and only if the original graph has treewidth at most two. Now we solve the problem on the new connected graph with the reduction system given in the previous section. After that, we remove the dummy vertex from all nodes it occurs in, and the resulting tree decomposition is a tree decomposition of width at most two of the input graph. For more details, see [9].

Theorem 4.1. *There is a parallel algorithm which checks whether a given (*B*-labeled) graph G has treewidth at most two, and if so, returns a tree decomposition of width at most two of G. The algorithm uses* $O(n+m)$ *operations and space, and* $O(\log(n+m))$ *time on a CRCW PRAM, or* $O(\log(n+m)\log^*(n+m))$ *time on an EREW PRAM.*

If the input graph $G = (V,E)$ is simple, then we can use the same preprocessing step as described in [8] (Section 5) for series parallel graphs. This results in the following.

Theorem 4.2. *There is a parallel algorithm which checks whether a given simple graph G has treewidth at most two, and if so, returns a tree decomposition of width at most two of G. The algorithm uses* $O(n)$ *operations and space, and* $O(\log n)$ *time on a CRCW PRAM, or* $O(\log n \log^* n)$ *time on an EREW PRAM.*

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 [3, 10, 9]. With the results of this paper, this implies that we can solve the problems described above on graphs of treewidth at most two with the same resource bounds as the algorithm for finding a tree decomposition of width two.

One of the problems which can be solved if a tree decomposition of bounded width of the input graph is given, is the pathwidth problem: given a graph G and an integer constant k, check whether G has pathwidth at most k, and if so, find a path decomposition of width at most k of the graph [9]. Hence we have the following result.

Theorem 4.3. *Let* $k \geq 1$ *be an integer constant. There is a parallel algorithm which checks whether a given simple graph G has treewidth at most two and pathwidth at most k, and if so, returns a path decomposition of width at most k of G. The algorithm uses* $O(n)$ *operations and space, and* $O(\log n)$ *time on a CRCW PRAM, or* $O(\log n \log^* n)$ *time on an EREW PRAM.*

Note that the theorem also holds for multigraphs, if we replace n by $n+m$ in the time and operations bounds. As graphs of pathwidth at most two also have treewidth at most two, the theorem implies that we can find a path decomposition of width at most two of a graph, if one exists, within the same resource bounds.

Acknowledgement

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

References

[1] K. R. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka. A simple parallel tree contraction algorithm. *J. Algorithms*, 10:287–302, 1989.

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

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

[4] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Alg. Disc. Meth.*, 7:305–314, 1986.

[5] H. L. Bodlaender. NC-algorithms for graphs with small treewidth. In J. van Leeuwen, editor, *Proceedings 14th International Workshop on Graph-Theoretic Concepts in Computer Science WG'88*, pages 1–10. Springer Verlag, Lecture Notes in Computer Science, vol. 344, 1988.

[6] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[7] 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.

[8] H. L. Bodlaender and B. de Fluiter. Parallel algorithms for series parallel graphs. Technical Report UU-CS-1997-21, Dept. of Computer Science, Utrecht University, Utrecht, the Netherlands, 1997.

[9] 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. To appear in SIAM J. Computing, 1997.

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

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

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

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

[14] D. Granot and D. Skorin-Kapov. NC algorithms for recognizing partial 2-trees and 3-trees. *SIAM J. Disc. Meth.*, 4(3):342–354, 1991.

[15] J. Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *J. Algorithms*, 20:20–44, 1996.

[16] J. Matoušek and R. Thomas. Algorithms finding tree-decompositions of graphs. *J. Algorithms*, 12:1–22, 1991.

[17] D. P. Sanders. On linear recognition of tree-width at most four. *SIAM J. Disc. Meth.*, 9(1):101–117, 1996.