Contractions, Removals and How to Certify 3-Connectivity in Linear Time

Jens M. Schmidt
Max Planck Institute for Informatics

Abstract

One of the most noted construction methods of 3-vertex-connected graphs is due to Tutte and based on the following fact: Any 3-vertex-connected graph $G = (V, E)$ on more than 4 vertices contains a contractible edge, i. e., an edge whose contraction generates a 3-connected graph. This implies the existence of a sequence of edge contractions from G to the complete graph K_4, such that every intermediate graph is 3-vertex-connected. A theorem of Barnette and Grünbaum gives a similar sequence using removals on edges instead of contractions.

We show how to compute both sequences in optimal time, improving the previously best known running times of $O(|V|^2)$ to $O(|E|)$. This result has a number of consequences; an important one is a new linear-time test of 3-connectivity that is certifying; finding such an algorithm has been a major open problem in the design of certifying algorithms in the last years. The test is conceptually different from well-known linear-time 3-connectivity tests and uses a certificate that is easy to verify in time $O(|E|)$. We show how to extend the results to an optimal certifying test of 3-edge-connectivity.

1 Introduction

The class of 3-connected (i. e., 3-vertex-connected) graphs has been studied intensively for many reasons in the past 50 years. Algorithmic applications include problems in graph drawing (see [42] for a survey), problems related to planarity [6, 28] and online problems on planar graphs (see [14] for a survey). From a complexity point of view, 3-connectivity is in particular important for problems dealing with longest paths, because it lies, somewhat surprisingly, on the borderline of NP-hardness: Finding a Hamiltonian cycle is NP-hard for 3-connected planar graphs [25] but becomes solvable in linear running time [11] for higher connectivity, as 4-connected planar graphs are Hamiltonian [63].

From a top-level view, we design an efficient algorithm from an *inductively defined construction* of a graph class. For a given graph class $\mathcal{C}$, such constructions start with a set of base graphs and apply iteratively operations from a finite set of operations such that precisely the members of $\mathcal{C}$ are constructed. This does not only give a computational approach to test graphs for membership in $\mathcal{C}$, it can also be exploited to prove properties of $\mathcal{C}$ using just arguments on the base graphs and the finitely many operations. Graph theory provides inductively defined constructions for many graph classes, including planar graphs, triangulations, k-connected graphs for $k \leq 4$, regular graphs and various intersections of these classes [4, 5, 32]. Most of these constructions have not been exploited computationally.

This research was partially supported by the Deutsche Forschungsgemeinschaft within the research training group "Methods for Discrete Structures" (GRK 1408) and partly done while at FU Berlin.

For an inductively defined construction of $\mathcal{C}$ and a graph $G \in \mathcal{C}$, we call a sequence of operations that is applied to a specified base graph to construct G a *construction sequence* of G. We will also identify a construction sequence with the sequence of graphs it constructs, provided that this determines the construction sequence uniquely. Sometimes, it is more convenient to describe a construction sequence by giving the inverse operations from G to a specified base graph; in such cases, we refer to them as *top-down* variants of construction sequences, as opposed to *bottom-up* variants.

One of the most noted constructions for 3-connected graphs was given by Tutte [64] and is based on the following fact: Any 3-connected graph G on more than 4 vertices contains a *contractible* edge, i. e., an edge whose contraction generates a 3-connected graph. Iteratively contracting such an edge gives a top-down construction sequence from G to a K_4-multigraph (i. e., a K_4 with possible additional parallel edges and self-loops), in which all intermediate graphs are 3-connected. In general, also non-3-connected graphs can contain contractible edges, but adding an additional condition establishes a full characterization: A graph G on more than 4 vertices is 3-connected if and only if there is a construction sequence from G to a K_4-multigraph that uses only contractions on edges with both end vertices having at least 3 neighbors [17]; we will call this a *sequence of contractions*. It is also possible to describe a sequence of contractions bottom-up by using the inverse operations edge addition and vertex splitting; in fact, this is the original form as stated in Tutte's famous wheel theorem [64].

Barnette and Grünbaum [3] and Tutte (implicitly shown in Theorems 12.64 and 12.65 of [65]) give a different construction of 3-connected graphs that is based on the following argument: Any 3-connected graph G on more than 4 vertices contains a *removable* edge, i. e., an edge whose deletion generates a subdivision of a 3-connected graph. *Removing* this edge leads, similar as in the sequence of contractions, to a top-down construction sequence from G to K_4. We will define *removals* in Section 3 and show how to add an additional condition to fully characterize 3-connected graphs. Again, the originally proposed construction is given bottom-up from K_4 to G, using three operations.

Although both existence theorems on contractible and removable edges are used frequently in graph theory [57, 58, 65], the first non-trivial computational result to create the corresponding construction sequences was published more than 45 years afterwards: In 2006, Albroscheit [1] gave an algorithm that computes a construction sequence for 3-connected graphs in $O(|V|^2)$. However, in this algorithm, contractions and removals are allowed to intermix.

In 2010, two algorithms of the same running time $O(|V|^2)$ were given [40, 47] that both compute a sequence of contractions. The latter algorithm computes alternatively a construction sequence that uses only removals in $O(|V|^2)$. One of the building blocks of this algorithm is a straight-forward transformation from an arbitrary sequence of removals to a sequence of contractions in time $O(|E|)$. This shows that the sequence of Barnette and Grünbaum is algorithmically at least as powerful as the sequence of contractions. All mentioned algorithms do not rely on the 3-connectivity test of Hopcroft and Tarjan [30], which runs in linear time but is rather involved. No algorithm that computes any of these sequences in subquadratic time is known; in particular, there is no obvious way of improving the $O(|V|^2)$ algorithms to linear time.

The main contribution of this paper is an algorithm that computes the construction sequence of Barnette and Grünbaum bottom-up in time and space $O(|E|)$. The key idea is based on a careful classification and grouping of certain paths in a simple decomposition of the graph; the groups of paths can be decomposed later into the desired operations for Barnette and Grünbaum's

construction. The result has a number of consequences and applications.

Top-down and bottom-up variants of both constructions. One can immediately obtain the sequence of removals from Barnette and Grünbaum's bottom-up construction sequence by replacing every operation with its inverse removal operation. Applying the transformation of [47] will imply optimal time and space algorithms for the sequence of contractions, its bottom-up variant and related sequences as well.

Certifying 3-connectivity in linear time. Blum and Kannan [7] initiated the concept of programs that check their work. Mehlhorn and Näher [33, 38, 39] (see [36] for an extensive survey) introduced the concept of *certifying algorithms*, i. e., algorithms that produce with each output a *certificate* that the particular output has not been compromised by a bug. Such a *certificate* can be any data that allows to check the correctness of the particular output (uniformly using a verifying algorithm), having only access to the input.

Developing certifying algorithms is a major goal for problems where the fastest solutions known are complicated, as implementations of these solutions tend to be error-prone. Having a certifying algorithm for such a problem allows to verify the output of *every particular* instance, regardless of how complicated the computation of this output is. For this reason, certifying algorithms are these days used in software libraries like LEDA [38] for reliability purposes.

A prominent example for a problem for which the fastest solutions known are complicated is testing a graph for 3-connectivity. Although sophisticated linear-time recognition algorithms are known for over 35 years [21, 30, 46, 67, 68], none of them describe an easy-to-verify certificate or show how a construction sequence can be computed.

The currently fastest algorithms that certify 3-connectivity need $O(|V|^2)$ time and use construction sequences as certificates [1, 40, 47]. Recently, based on the results in [18], a linear time certifying algorithm for 3-connectivity has been given for the subclass of Hamiltonian graphs, when a Hamiltonian cycle is given as part of the input [17]. In general, finding a certifying algorithm for 3-connectivity in subquadratic time is an open problem [36, Chapter 5.4, p. 26] [17].

We solve this problem by giving a linear-time certifying algorithm for 3-connectivity that uses Barnette and Grünbaum's construction sequence as certificate. This implies a new linear-time test for 3-connectivity that neither assumes the graph to be 2-connected nor needs to compute low-points (see [30] for a definition of low-points); instead, it uses the structure of 3-connected graphs implicitly by applying simple path-generating rules. This is conceptually different from all previous linear-time 3-connectivity tests. The algorithm has already been implemented and made publicly available [44]; interestingly, it outperforms the test in [30] on no-instances.

In addition, we give a simple decomposition of graphs that is closely related to ear decompositions [34, 69] and which allows to unify existing tests on 2-connectivity [9, 16, 19, 20, 22, 53, 55] with tests on 2-edge-connectivity [54, 59, 62].

Certifying 3-edge-connectivity in linear time. There is no test for 3-edge-connectivity that is certifying and runs in linear time, although many non-certifying linear-time algorithms for this problem are known [23, 43, 52, 60, 61]. The first (non-certifying) one due to Galil and Italiano [23] uses the fact that testing a graph G on k-edge-connectivity can be reduced to testing k-vertex-connectivity on a slightly modified graph. Based on this reduction, we give a linear-time test on 3-edge-connectivity that is certifying.

Applications. Being able to certify 3-connectivity efficiently has a number of applications. E. g., many graph algorithms [13, 14, 6, 28, 42] that use the *SPQR-tree* data structure [27] can be made certifying. Due to the Theorem of Steinitz [51], algorithms on *polytopes* (in a graph representation) can be augmented with a quick and easy check that their input represents indeed a polytope. Applications in communication networks include certificates for their *reliability* (depending on 2- or 3-connectivity) and the property to allow for a *perfectly secure message transmission* [15].

We first develop results about construction sequences in Section 3. In Section 4, we discuss certificates for k-connectivity and k-edge-connectivity, $0 \leq k \leq 3$, and show how these can be certified by conceptually simple checkers. Section 4 also introduces a decomposition of the input graph, which partitions its edge set into cycles and paths, both of which are called *chains*. On a high level view, chains provide the structure that we need to compute the next operation of the construction sequence efficiently. Section 5 shows how to compute a construction sequence of a 3-connected graph in linear time. Section 5.7.2 describes how to find a cut vertex or separation pair when the input graph is not 3-connected.

2 Preliminaries

We consider finite undirected graphs G with $n := |V(G)|$ vertices and $m := |E(G)|$ edges and use standard graph-theoretic terminology from [8], unless stated otherwise. A set of vertices (respectively, of edges) that leaves a disconnected graph upon deletion is called a *vertex cut* (respectively, an *edge cut*). Vertex cuts of size one and two are called *cut vertices* and *separation pairs*, respectively; edge cuts of size one in connected graphs are called *bridges*. For $k \geq 1$, let a graph G be *k-connected* if $n > k$ and there is no vertex cut X in G with $|X| < k$. The *connectivity* of a graph G is the largest integer k such that G is k-connected. Note that k-connectivity neither depends on parallel edges nor on self-loops. For $k \geq 1$, let a graph G be *k-edge-connected* if $n > 1$ and there is no edge cut X in G with $|X| < k$.

Let $v \to_G w$ denote a path P from vertex v to vertex w in G and let $s(P) = v$ and $t(P) = w$ be the source and target vertex of P, respectively (as G is undirected, the direction of P is given by $s(P)$ and $t(P)$). Every vertex in $P \setminus \{s(P), t(P)\}$ is called an *inner vertex* of P. For a vertex v in G, let $N(v) = \{w \mid vw \in E\}$ denote its set of neighbors and $deg(v)$ its degree (counting multiedges). Let $\delta(G)$ be the minimum degree in G.

Let T be an undirected tree rooted at vertex r. For two vertices x and y in T, let x be an *ancestor* of y and y be a *descendant* of x if $x \in V(r \to_T y)$. If additionally $x \neq y$, x is a *proper* ancestor and y is a *proper* descendant. For a vertex x in T, let $T(x)$ be the subtree of T that contains exactly the descendants of x. A *DFS-forest* F of a graph G is an acyclic graph that is computed by performing a depth-first search (DFS), see [53]. The edges in $E(G) \setminus E(F)$ are called *backedges*. Every connected component T in F is a rooted tree, which is called *DFS-tree* of G.

Let K_2^m be the graph on 2 vertices that contains exactly m parallel edges and no self-loops. A decomposition of G is a set of subgraphs of G whose edge-sets partition $E(G)$.

3 Construction Sequences

Let an edge e in a graph be *contractible* if contracting e results in a 3-connected graph. A *subdivision* of a graph G is a graph generated from G by replacing each edge of G by a path of length at least

one. Conversely, we want a notation to get back to the graph without subdivided edges. Let *smoothing* a vertex $v \in V(G)$ be the operation on G that, if $deg(v) = 2$ and $|N(v)| = 2$ (both not counting self-loops), deletes v followed by adding an edge between its neighbors.

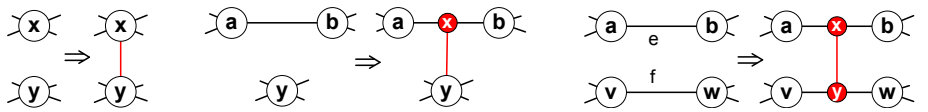
Removing an edge $e = xy$ (with $x \neq y$) in a graph deletes e followed by smoothing x and y. An edge of G that is not a self-loop is called *removable*, if removing it generates a 3-connected graph. Iteratively removing removable edges in a 3-connected graph G leads to a *sequence of removals* from G to K_4, the existence of which characterizes 3-connected graphs when adding an additional condition similar as for the sequence of contractions. We describe the equivalent bottom-up construction due to Barnette and Grünbaum.

Definition 1. The following operations are defined on 3-connected graphs and called *BG-operations* (see Figure 1).

(a) Add an edge xy with $x \neq y$ (possibly a parallel edge).

(b) Subdivide an edge ab, $a \neq b$, by a vertex x and add the edge xy for a vertex $y \notin \{a, b\}$.

(c) Subdivide two non-parallel edges by vertices x and y, respectively, and add the edge xy.



(a) possibly a parallel edge

(b) y, a and b are pairwise distinct

(c) e and f are neither identical nor parallel

Figure 1: The three BG-operations.

A classical result of Barnette and Grünbaum [3] and Tutte [65] states that BG-operations characterize the 3-connected graphs: A graph G is 3-connected if and only if G can be constructed from K_4 using BG-operations. Therefore, if G is 3-connected, every intermediate graph obtained by performing the previous construction must be 3-connected as well.

Corollary 2 (Barnette and Grünbaum [3], Theorems IV.14 – IV.18 in Tutte [66])**.** Applying a BG-operation to a 3-connected graph G generates a 3-connected graph.

A BG-operation can always be inversed by removing a removable edge. Figure 2 shows that the converse is not true. We show under which conditions the inverse operation of removing a removable edge is a BG-operation.

Lemma 3. *The inverse operation of removing a removable edge $e = xy$ in a graph G is a BG-operation if and only if either $|V(G)| = 4$ or $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$ (in G).*

Proof. Let G' be the 3-connected graph generated from G by removing e and let O be the inverse operation of this removal. Assume first that O is a BG-operation and assume further that $|V(G)| > 4$. With Corollary 2, G is 3-connected, which implies $|N(x)| \geq 3$ and $|N(y)| \geq 3$ in G. It remains to show that $|N(x) \cup |N(y)| \geq 5$. Assume the contrary. Then $|N(x)| = |N(y)| = 3$, as $|N(x)| \geq 4$ or $|N(y)| \geq 4$ would imply $|N(x) \cup |N(y)| \geq 5$, as x and y are adjacent. For the same reason,

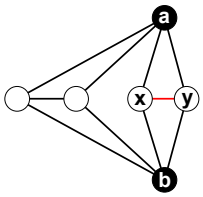


Figure 2: The edge xy is removable, since removing xy results in a K_4-multigraph, which is 3-connected. However, the inverse operation of removing xy is not a BG-operation, because it induces the separation pair $\{a, b\}$.

$|N(x) \cap N(y)| = 2$. Since $|V(G)| > 4$, some vertex in $N(x) \cap N(y)$ must be adjacent to a vertex that is neither adjacent to x nor y. Then $N(x) \cap N(y)$ is a separation pair, which contradicts the 3-connectivity of G. This proves $|N(x) \cup |N(y)| \geq 5$ in G.

Assume that O is not a BG-operation. It suffices to show that $|V(G)| \neq 4$ and $|N(x)| < 3$, $|N(y)| < 3$ or $|N(x) \cup N(y)| < 5$. We distinguish cases by the number of end vertices of e that are deleted in the process of removing e. This number must be either 0, 1 or 2. It cannot be 0, as in that case the removal just deletes e, implying that O just adds an edge, which would be a BG-operation. Therefore, G contains more vertices than the 3-connected graph G'. Since the smallest 3-connected graph is K_4, $|V(G)| \neq 4$.

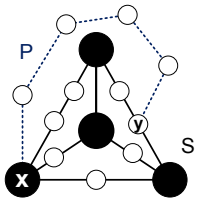
If exactly one vertex, say x, is deleted by the removal, let a and b be the two neighbors of x in G that are different from y. Then $y \in \{a, b\}$ holds to assure that O is not a BG-operation and $|N(x)| < 3$ in G. If both vertices x and y are deleted by the removal, let f_1 and f_2 be the edges in which x and y are smoothed, respectively. Then f_1 and f_2 have to be identical or parallel in order to assure that O is not a BG-operation, which implies that $|N(x) \cup N(y)| < 5$. □

If G is simple, the condition $|V(G)| = 4$ in Lemma 3 is not needed.

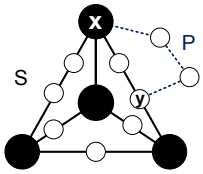
Lemma 3 implies that a graph G is 3-connected if and only if a sequence of removals on removable edges $e = xy$ with $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$ to a K_4-multigraph exists. The following alternative view on this sequence allows to represent intermediate graphs as subgraphs of G. Let a vertex v in a subdivision S of either K_2^3 or of a 3-connected graph be *real* if $deg(v) \geq 3$. Let $V_{real}(S)$ be the set of all real vertices in S. Let the *links* of S be the paths in S that have real end vertices but contain no other real vertices. The links of S are in one-to-one correspondence to the edges of the subdivided graph and therefore partition $E(S)$. Let two links be *parallel* if they share the same end vertices.

Definition 4. Let S be a subgraph of a graph G such that S is a subdivision of either K_2^3 or of a 3-connected graph. A *BG-path* for S is a path $P = x \rightarrow_G y$, $x \neq y$, with the properties (see Figure 3):

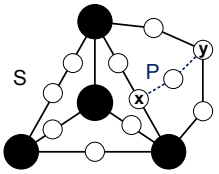
1. $V(P) \cap V(S) = \{x, y\}$.
2. Every link of S that contains both x and y contains them as end vertices.
3. If x and y are inner vertices of distinct links L_x and L_y of S, respectively, and $|V_{real}(S)| \geq 4$, then L_x and L_y are not parallel.



(a) P is a BG-path for the K_4-subdivision S.



(b) P is not a BG-path for S, since there is a link that contains x and y such that y is not an end vertex of that link (see Property 4.2).



(c) P is not a BG-path for the K_4-multigraph S, as x and y are inner vertices of two parallel links and $|V_{real}(S)| \geq 4$ (see Property 4.3).

Figure 3: BG-paths

Theorem 5 (implicitly in [3])**.** *A graph G without self-loops is 3-connected if and only if $\delta(G) \geq 3$ and G can be constructed from a K_4-subdivision in G by adding BG-paths.*

Theorem 5 implies that every 3-connected graph contains a subdivision of K_4, a result first shown by J. Isbell [3]. Note that after the addition of a BG-path P, $s(P)$ and $t(P)$ are real. Let $S_4, \ldots, S_z$ be a sequence of subgraphs in G that are generated by the construction, i. e., S_4 is the K_4-subdivision in G and $S_z = G$. The BG-path construction has the advantage that every S_l is a subgraph of $S_{l+1} \subseteq G$. This admits not only a straight-forward representation of the construction in linear space by storing S_4 and the added BG-paths, it provides also a way to compute a next BG-path efficiently by searching the neighborhood of the current subgraph in G.

The choice of the K_4-subdivision S_4 is not crucial [47]: At the expense of having additional parallel links in subgraphs S_l (we will be able to handle these efficiently), there exists a construction sequence to a 3-connected graph G using BG-paths from *every* prescribed K_4-subdivision in G. To certify 3-connectivity, we will use a slightly modified variant of the BG-path construction that starts with a K_2^3-subdivision instead of a K_4-subdivision; let the sequence of generated subgraphs be in that case $S_3, \ldots, S_z$. We will provide linear-time transformations to all other construction sequences.

We summarize constructive characterizations of 3-connected graphs and group similar ones. From now on, we assume G to be simple, although all results extend to multigraphs. When convenient, a construction sequence of a special type A is referred to as *sequence A*, although there might be more than one sequence of that type.

Theorem 6. *A simple graph G is 3-connected*

$\Leftrightarrow$ *There is a sequence of BG-operations from K_4 to G* (1)
(see [3], Theorems 12.64 and 12.65 in [65])

$\Leftrightarrow$ *There is a sequence of BG-operations from K_4 to G such that every* (2)
intermediate graph is simple (see [3], Theorem 6.(6) in [48])

$\Leftrightarrow$ *There is a sequence of removals from G to K_4 of* removable *edges $e = xy$ with* (3)
$|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$ (see Lemma 3)

$\Leftrightarrow$ *There is a sequence of removals from G to K_4 of edges $e = xy$ with* (4)
$|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$

$\Leftrightarrow$ *There is a sequence of contractions from G to a K_4-multigraph of* contractible *edges $e = xy$ with $|N(x)| \geq 3$ and $|N(y)| \geq 3$ (see [47], chapter 5 in [64])* (5)

$\Leftrightarrow$ *There is a sequence of contractions from G to a K_4-multigraph of edges $e = xy$ with $|N(x)| \geq 3$ and $|N(y)| \geq 3$ (see [17])* (6)

$\Leftrightarrow$ *$\delta(G) \geq 3$ and there is a sequence of BG-paths from a K_2^3-subdivision in G to G such that the first BG-path generates a K_4-subdivision* (7)

$\Leftrightarrow$ *$\delta(G) \geq 3$ and there is a sequence of BG-paths from a K_4-subdivision in G to G [3, 47]* (8)

$\Leftrightarrow$ *$\delta(G) \geq 3$ and there is a sequence of BG-paths from* each *K_4-subdivision in G to G [47]* (9)

Proof. Only the equivalences of (4) and (7) need to be shown. We first prove (7) $\Leftrightarrow$ (8). Clearly, deleting the first BG-path in a sequence (7) gives a sequence (8). Conversely, the K_4-subdivision of a sequence (8) can be decomposed into a K_2^3-subdivision S_3 and a path P that connects inner vertices of two parallel links of S_3. Since $|V_{real}(S_3)| < 4$, P is a BG-path for S_3.

For the equivalence of (4), clearly (3) $\Rightarrow$ (4). We prove (4) $\Rightarrow$ (1). Let a sequence (4) be given and let G_{i+1} be any graph in that sequence different from K_4. Let xy be the edge in G_{i+1} that is removed in the sequence to generate the graph G_i. Assume first that G_i is 3-connected. Then xy is removable and since $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$ hold in G_{i+1}, the inverse of removing xy is a BG-operation on G_i due to Lemma 3. In particular, the 3-connectivity of G_i implies that G_{i+1} is 3-connected with Corollary 2. Since K_4 is 3-connected, the inverse of every removal in the sequence is a BG-operation, giving a sequence (1). □

We list linear-time transformations between the sequences of Theorem 6.

Lemma 7 ([47], Lemmas 7 and 13 in [48])**.** *There are algorithms that transform every given sequence (1)–(4) and (8) to each of the sequences (1) and (3)–(8) in linear time. The transformations to sequence (1), (3), (4) and (8), respectively, preserve the number of operations.*

Lemma 8. *Every sequence (1) can be transformed to a sequence (2) in linear time such that the number of operations is preserved.*

Proof. All vertices and edges in sequence (1) have unique identifiers. Whenever an edge is subdivided by a BG-operation, new identifiers are assigned to the new vertex and to the two parts of the former edge, respectively. Similarly, a new identifier is assigned to the *newly created* edge of every BG-operation, i.e., to the edge xy in Definition 1. Once created, vertices do not change their identifiers. In [47], it is shown that this representation can be stored in linear space.

Let F be the list of all edges that are contained in this representation with different identifiers, ordered by their time of creation in the construction sequence. F contains the edges that are newly created by BG-operations, the edges that arise from subdividing old edges and the edges that are contained in the initial K_4 (the latter ones are placed at the very beginning of F).

We first show how to group edges of F that have the same end vertices. Let $<$ be any total ordering on the vertices. We sort the edges of F in lexicographic order $\prec$ on their end vertices, i.e., for two distinct edges ab and cd in F with $a < b$ and $c < d$, $ab \prec cd$ if either $a < c$ or $a = c$ and $b < d$. This sorting can be computed in linear time by performing two bucket sorts on the end vertices of edges in F, respectively.

Sorting F allows us to efficiently group edges that have the same end vertices. Let $F_{a,b}$ be the list of the edges in F with end vertices a and b; for every edge e in $F_{a,b}$, we store pointers to $F_{a,b}$

and to the edge $sub(e)$ that subdivides e (if exists), respectively. Since bucket sort is stable, the edges in $F_{a,b}$ are still ordered by their time of creation in the construction sequence.

In order to make every intermediate graph simple, we swap edges in $F_{a,b}$ and move the operations that create some of these edges to another position in the construction sequence. Clearly, we only have to consider lists $F_{a,b}$ that contain at least two edges, as otherwise no parallel edge ab occurs in the construction sequence. Thus, $|F_{a,b}| \geq 2$. There is at most one edge in $F_{a,b}$ that is not subdivided, since G is simple. Hence, $F_{a,b}$ contains at least one edge e for which $sub(e)$ exists.

Let $e_{\min}$ be the edge in $F_{a,b} = \{e_1, \ldots, e_k\}$ that is subdivided earliest in the construction sequence. This edge can be computed in time $O(k)$ by passing once through the list $F_{a,b}$. If $e_{\min} \neq e_1$, we swap $e_{\min}$ and e_1 throughout the construction sequence. This modification preserves every operation to be a BG-operation, as the vertices a and b are never deleted (recall that adding an edge between two existing vertices is always a BG-operation) and since no edge in $F_{a,b}$ is subdivided before $e_{\min}$. In fact, all parallel edges ab are pairwise interchangeable until $sub(e_{\min})$ is added. Clearly, the number of BG-operations has not changed.

Every remaining edge $e \in \{e_2, \ldots, e_k\}$ in $F_{a,b}$ is now the newly added edge of some BG-operation O_e, as a and b exist due to e_1. Note that e_1 is not necessarily the newly added edge of some BG-operation; e.g., e_1 may be part of the initial K_4 instead. If there is an edge e in $F_{a,b}$ that is not subdivided, e must be distinct from e_1 because of the previous modification. If such an edge e exists, we move it to the very end of the construction sequence. This preserves any operation to be a BG-operation, as a and b exist and no other edge interferes with e, as no edge subdivides e.

For every remaining edge $e \in F_{a,b}$, we move O_e to the position in the construction sequence immediately before $sub(e)$ is added. Again, this preserves every operation in the construction sequence to be a BG-operation, as a and b exist and $sub(e)$ is the only edge that interferes with e. Since every such e is now added after $e_{\min}$ in the construction sequence and immediately subdivided after its addition, every intermediate graph is simple. Clearly, all modifications preserve the number of BG-operations. □

A sequence (7) can be easily transformed to a sequence (8) by using the first BG-path to create a K_4-subdivision; the transformed sequence will have one operation less. Moreover, Lemmas 7 and 8 allow to further transform a sequence (8) efficiently to every sequence (1)–(6). In the rest of the paper we will therefore focus on computing only a sequence (7).

That is why a *construction sequence* will refer to a sequence (7) throughout the rest of the paper: We can transform this sequence to every other sequence of Theorem 6 in linear time. The following lemma provides an iterative algorithmic approach to build a sequence (7).

Lemma 9 (Corollary of [48])**.** *Let G be a 3-connected graph and $H \subset G$ with H being a subdivision of either K_2^3 or of a 3-connected graph. Then there is a BG-path for H in G. Moreover, every link L of H of length at least 2 has a parallel link (maybe L itself) that contains an inner vertex on which a BG-path for H starts.*

Every sequence of contractions (i. e., every sequence (5) and (6)) contains exactly $n-4$ contractions, implying that the K_4-multigraph contains exactly $m-n-2$ parallel edges. The next lemma is a corresponding result for the number of operations in construction sequences.

Lemma 10. *Every sequence (1)–(4) and (8) contains exactly $m-n-2$ operations, whereas every sequence (7) contains exactly $m-n-1$ operations.*

Proof. It suffices to show the first claim for each sequence (1), as Lemmas 7 and 8 preserve the number of operations, respectively. The remaining claim follows directly from the fact that the transformation of a sequence (8) to a sequence (7) adds exactly one BG-path. Let a, b and c denote the number of BG-operations in a sequence (1) that create zero, one and two new vertices, respectively. Then $b + 2c = n - 4$ and $a + 2b + 3c = m - 6$ hold, since K_4 consists of four vertices and six edges. Subtracting the first from the second equation gives $a + b + c = m - n - 2$. □

4 Chain Decompositions and Certificates

Suppose there is a vertex or edge cut X of size $k - 1$, $k \geq 1$, in a graph G with $n > 1$ (for vertex-connectivity, let $n > k$). Then X would be a straight-forward certificate for G being not k-connected respectively not k-edge-connected. However, certificates should be as easy to check as possible, while the running time for computing the certificate is less important. We therefore apply the paradigm of *shifting as much as possible of the checker's work to the computation of the certificate*.

Instead of using X as certificate, we color the vertices of one connected component of $G \setminus X$ red and the vertices of all other connected components of $G \setminus X$ green. A checker for G being not k-connected then just needs to check that at most $k - 1$ vertices are uncolored, there is at least one red and one green vertex and that no edge joins a red vertex with a green one. For G being not k-edge-connected, it suffices to check that there is at least one red, one green and no uncolored vertex and that the end vertices of at most $k - 1$ edges differ in color. Note that this certifies a graph to be disconnected for $k = 1$. We will always use these certificates instead of using X itself. The certificates need space $O(n)$ and can be checked in time $O(m)$, as $n > m + 1$ proves G to be disconnected.

For a certifying algorithm, it remains to show how the vertex and edge cuts X for these certificates are computed and which certificates is to be used for k-connectivity and k-edge-connectivity, $1 \leq k \leq 3$. Performing a depth-first search [35, 53, 56] or any other suited graph traversal gives the connected components of G in linear time. A certificate for G being connected is given in [36], using an easy numbering scheme on the vertices.

Chain Decomposition. We use a very simple decomposition of graphs into cycles and paths, which links DFS-trees, ear decompositions [34] and open (i. e., no ear is a cycle) ear decompositions [34, 69] without needing to compute low-points in advance (see [31] for a definition of low-points). This decomposition does not only unify existing linear-time tests on 2-connectivity [9, 10, 16, 19, 20, 22, 53, 55] and 2-edge-connectivity [54, 59, 62], it will also be the base structure that allows to compute a construction sequence of a 3-connected graph efficiently. We define the decomposition algorithmically; a similar procedure for the computation of so-called low-points can be found in [46].

Let G be a simple but not necessarily connected graph and let T be a depth-first search forest of G. The DFS assigns a depth-first index (DFI) to every vertex. Recall that, for every backedge e, $s(e)$ and $t(e)$ are the two end vertices of e such that $s(e)$ is a proper ancestor of $t(e)$ in T.

We decompose G into a set $C = \{C_1, \ldots, C_{|C|}\}$ of cycles and paths, called *chains*, by applying the following procedure for each vertex v in ascending DFI-order: Let T' be the tree in the DFS-forest T that contains v and let r be the root of T'. For every backedge vw with $s(vw) = v$, we

traverse the path $w \to_{T'} r$ until a vertex x is found that is either r or already contained in a chain. The traversed subgraph $vw \cup (w \to_{T'} x)$ forms a new *chain* C_i with $s(C_i) = v$ and $t(C_i) = x$.

We call C a *chain decomposition* (although it does not necessarily partition $E(G)$ when G is not 2-edge-connected). Let $<$ be the strict total order on C in which the chains were found, i. e., $C_1 < \cdots < C_{|C|}$, when forming the decomposition. Processing the vertices in ascending DFI-order is not crucial; any pre-order on $V(T)$ can be used instead (a strict total order $\prec$ on $V(T)$ is a *pre-order* if, for every vertex $v \in V(T)$, all proper ancestors of v in T precede v in $\prec$). Clearly, the decomposition into chains can be computed in time $O(n + m)$.

Lemma 11. *Let C be a chain decomposition of a simple graph G. Then G is 2-connected if and only if $\delta(G) \geq 2$ and C_1 is the only cycle in C.*

Proof. Let G be 2-connected. Then G contains at least 3 vertices, the DFS-forest T is a tree and the root r of T has exactly one child, as otherwise r would be a cut vertex. Additionally, G cannot contain a vertex of degree one, as otherwise its neighbor would be a cut vertex. It follows that r is adjacent to a backedge, implying that C_1 is a cycle. Assume to the contrary that another chain $C_i \neq C_1$ is a cycle. Let v be the vertex in C_i of minimal DFI and let w be the child of v in T that is also in C_i. Then no backedge adjacent to a vertex with smaller DFI than v can enter $T(w)$, as this backedge would have been processed before in the chain decomposition, contradicting C_i. Because $v \neq r$, v is a cut vertex, which contradicts the assumption and the claim follows.

Now let every vertex in G have degree at least two and let C_1 be the only cycle in C. This implies that G is connected, thus, T is a tree. Assume to the contrary that G is not 2-connected and consider the decomposition of G into maximal 2-connected components (*2-components*), where single edges are regarded as 2-connected graphs. It is well-known that representing each 2-component by a vertex that is adjacent to every cut vertex contained in that 2-component gives a tree [24, 29], called the *block-cut-tree* (*BC-tree*). Every cycle in G must be contained in a 2-component, as a cycle is 2-connected. Let X be the 2-component that contains C_1, let r be the root of T and let Y be a leaf of the BC-tree that is different from X. Then Y is not an edge, as otherwise it would contain a vertex with degree one. It follows that Y contains a cycle. Let a be the last cut vertex on a path from r to an arbitrary vertex in Y (it may happen that $a = r$). Then the chain decomposition must find a cycle in Y when processing a. This cycle is different from $C_1 \subseteq X$, which contradicts the assumption. □

Lemma 12. *Let C be a chain decomposition of a simple graph G. Then G is 2-edge-connected if and only if G is connected and the chains in C partition $E(G)$.*

Proof. Let G be 2-edge-connected. In particular, G is connected and the DFS-forest T is a tree. By construction, the edge-sets of chains in C are disjoint and every backedge is contained in a chain. We show that every edge $e = xy$ in T (say, x is an ancestor of y) is also contained in a chain of C. There must be a cycle in G that contains e, as otherwise e would be a bridge. It follows that there is a backedge that enters $T(y)$. The chain in C containing the first such backedge that is traversed in the chain decomposition contains e.

Let G be not 2-edge-connected. We show that G is not connected or that the chains in C do not partition $E(G)$. Assume w. l. o. g. that G is connected, thus, T is a tree. Then G contains a bridge $e = xy$ (say, x is an ancestor of y), which must be a DFS-tree edge, as every backedge is contained in a cycle. As no backedge enters $T(y)$, e cannot be contained in any chain of C. □

4.1 Certificates for Low Connectivity

If G is connected, we can detect whether G is 2-connected or 2-edge-connected in linear time using Lemmas 11 and 12. If G is not 2-connected, we extract a cut vertex as follows: If a vertex with degree one exists, its neighbor is a cut vertex. Otherwise, a chain decomposition C contains a cycle $C_i \neq C_1$ according to Lemma 11. Then the vertex with minimal DFI in C_i is a cut vertex. If G is not 2-edge-connected, every edge that is not contained in a chain of C is a bridge.

For G being 2-connected and 2-edge-connected, C will be an open ear decomposition and an ear decomposition, respectively. It is well-known that these decompositions characterize the 2-connectivity respective 2-edge-connectivity of a graph [34, 69]. We therefore use in both cases C as a certificate that can be verified by going along the definition of (open) ear decompositions in time $O(m)$. Note that C is not an arbitrary (open) ear decomposition; it depends on the DFS-tree.

For 3-connectivity, we will use a sequence (7) as certificate, as this is a proof for 3-connectivity due to Theorem 6. A simple checker for a sequence (8) with running time $O(m)$ is given in [47] that can be easily extended to check a sequence (7) by replacing the K_4-subdivision by a K_2^3-subdivision. Moreover, this certificate can be easily stored in space $O(|E|)$ [47].

3-Edge-Connectivity. For 3-edge-connectivity, we use the reduction to 3-connectivity due to Galil and Italiano [23]. The reduction modifies the simple input graph G in linear time to a graph with $n + m$ vertices and $4m$ edges. First, a graph G' is generated from G by subdividing each edge with one vertex; these vertices are called *arc-vertices*. For each non-arc-vertex w in G' we do the following: Let $v_1, \ldots, v_{deg(w)}$ be the arc-vertices neighboring w in an arbitrary order. Then the edges $(v_1v_2, v_2v_3, \ldots, v_{deg(w)}v_1)$ are added to G' if not already existent. Note that the reduction blows up each vertex $v \in V(G)$ to a wheel graph with as many spokes as v has neighbors.

The graph G is 3-edge-connected if and only if G' is 3-connected [23]. Moreover, every vertex cut of minimal size in G' contains only arc-vertices (Lemma 2.2 in [23]). We apply the certifying 3-connectivity test to G' to obtain a certifying 3-edge-connectivity test in linear time and space. If G' is not 3-connected, the test for 3-connectivity returns a vertex cut of minimal size in G', which corresponds to an edge cut X of size at most two in G. The certificate then consists just of the red-green coloring of the connected components of $G \setminus X$ as described above.

Otherwise, G' is 3-connected and we get a sequence (7) for G'. The certificate consists of this sequence, G' and the injective mapping ϕ from each vertex in G' to its corresponding vertex or edge in G to certify the construction of G'. For a checker, it suffices to test that G' is 3-connected using the given sequence, every vertex in G has a unique preimage in $V(G')$ under ϕ, every non-arc-vertex v in G' is the hub of a wheel graph with $deg(v) + 1$ vertices that are all arc-vertices except for v, every two wheels in G' share at most one arc-vertex and every arc-vertex u in G' is incident to exactly two non-arc-vertices v and w such that $\phi(u) = \phi(v)\phi(w)$ and $\phi(u) \in E(G)$. Note that this checker may fail in detecting additional edges (but not additional vertices) in G and that this does not harm the 3-edge-connectivity of G. In both cases, the certificate needs linear space and can be checked in time $O(m)$.

5 A Certifying Algorithm in Linear Time

For convenience, we will assume that the input graph G is simple throughout this chapter. However, if needed, all results can be extended to non-simple graphs G by applying them to the underlying

simple graph of G; the underlying simple graph of G can be computed in time $O(n+m)$ by using two bucket sorts on $E(G)$.

We do not impose any other restrictions on G; in particular, we neither assume G to be 2-connected nor connected. The certifying algorithm that we will describe, actually tests a graph on having connectivity k for $k = 0$, 1, 2 and $k \geq 3$ and gives a certificate for each case. Its running time is $O(n+m)$. In the case that G is 3-connected, we will use a sequence (7) as certificate and show how to compute it in linear time. According to Lemmas 7 and 8, this implies that every sequence (1)–(8) of a 3-connected graph can be computed in time $O(n+m)$.

5.1 Using the Chain Decomposition

If $n \leq 1$, G has connectivity 0 and the number of vertices certifies that fact. Otherwise, we perform a DFS on G in time $O(n+m)$ and obtain a DFS-forest T. In particular, the DFS detects the connected components of G. If there is more than one connected component, we use the red-green coloring of Section 4 on the connected components as certificate that G is disconnected and, thus, has connectivity 0, as $n > 1$. For upcoming tests, we will assume that every vertex cut is certified by a red-green coloring.

In the remaining case, G is connected and T is a tree. If $n = 2$, G has connectivity 1. Otherwise, we compute a chain decomposition C on T in time $O(m)$ and obtain the chains $C_1 < \cdots < C_{|C|}$. Additionally, we compute the minimum degree $\delta(G)$ of G during the computation of the chain decomposition and store a vertex x that attains this degree. Moreover, by comparing the end vertices of each chain on identity, the number y of cycles in C is computed.

If $deg(x) = 1$, G is not 2-connected and the neighbor of x must be a cut vertex, which implies that G has connectivity 1. Let $deg(x) > 1$. If $y > 1$, the vertex with minimal DFI in a cycle $C_i \neq C_1$ is a cut vertex and can be computed directly from C in linear time; thus, G has connectivity 1. Otherwise, $y = 1$ and it follows that the root of T can have only one child. We conclude with $\delta(G) \geq 2$ that the only cycle in C is C_1. According to Lemma 11, G is 2-connected and C is an open ear decomposition, which is a certificate for the 2-connectivity of G.

If $n = 3$, G has connectivity 2. The same holds, if $deg(x) = 2$, as then the two neighbors of x form a separation pair. In the remaining case, G satisfies the following Property A.

Property A: $n \geq 4$, $\delta(G) \geq 3$ and G is 2-connected

From now on, we will assume Property A, as we dealt with all other cases. We summarize some implications.

Lemma 13. *The chains in C partition $E(G)$ and the last chain is C_{m-n+1}. The root r of T has exactly one child.*

Proof. According to Property A, G is 2-connected. As every 2-connected graph is 2-edge-connected, Lemma 12 implies that the chains in C partition $E(G)$. Since G is in particular connected, $|C| = m - n + 1$, as $|C|$ is the number of backedges in G by construction. If r had more than one child, the DFS-tree would imply that r is a cut vertex, contradicting the 2-connectivity of G. □

5.2 Computing a K_2^3-Subdivision

Assume for a moment that G is 3-connected. According to Lemma 9, it suffices to add iteratively BG-paths to an arbitrarily prescribed K_2^3-subdivision S_3 in G to get a construction sequence (7)

from S_3 to $S_z = G$. Note that we cannot make wrong decisions when choosing a BG-path, except for the first BG-path that has to generate a K_4-subdivision. The reason for this is that Lemma 9 can always be applied on the new generated subgraph and therefore ensures a completion of the sequence. With Lemma 10, $S_z = S_{m-n+2} = G$. Unfortunately, we do not know whether G is 3-connected. However, we can already compute a K_2^3-subdivision.

Let r be the root of T. Because of Property A, $deg(r) \geq 3$ in G. Since r has exactly one child in T, at least two chains exist and the chains C_1 and C_2 must both start at r. According to Lemma 11, C_1 is a cycle and all other chains are paths. By construction of the chains, $C_1 \cup C_2$ is a K_2^3-subdivision and we set $S_3 = C_1 \cup C_2$.

Recall that for a path $P = v \to_G w$, we defined $s(P) = v$ and $t(P) = w$. To keep notation simple, we abuse notation and split the cycle C_1 into the two different paths from r to $t(C_2)$, i. e., we set $C_0 = t(C_2) \to_T r$ and redefine $C_1 = r \to_{C_1 \setminus E(C_0)} t(C_2)$. Note that $s(C_0) = t(C_2)$ and $s(C_1) = r$. From now on, we will represent the chain decomposition as this list $C = C_0, \dots, C_{m-n+1}$ of paths. Sometimes, we will regard C as a set. By construction, the following holds.

Proposition 14. *Let C_i be a chain in $C \setminus \{C_0\}$. Then $s(C_i)$ is a proper ancestor of $t(C_i)$. Additionally, C_i contains exactly one backedge, namely its first edge.*

According to Lemma 12, every edge in G is contained in exactly one chain. We define parents and children of chains.

Definition 15. Let the *parent of a chain $C_i \neq C_0$* be the chain C_k that contains the edge from $t(C_i)$ to the parent of $t(C_i)$ in T. Conversely, let C_i be a *child* of the chain C_k.

The children of C_0 are exactly the chains C_i with $t(C_i) \in V(C_0)$. The children of a chain $C_k \neq C_0$ are exactly the chains C_i for which $t(C_i)$ is an inner vertex of C_k. The following two lemmas reveal much of the structure of chains and are often used in subsequent theorems.

Lemma 16. *Let $C_k \neq C_0$ be a chain with child C_i. Then $C_k < C_i$, $s(C_i)$ is a descendant of $s(C_k)$ in T and $t(C_i)$ is a proper descendant of $t(C_k)$ in T.*

Proof. By the definition of the parent relation, $t(C_i)$ must be an inner vertex of C_k and the last claim follows. As the traversal of C_i stopped at C_k in the chain decomposition, $C_k < C_i$ must hold. Therefore, $s(C_i)$ cannot be a proper ancestor of $s(C_k)$ in T. As T is a DFS-tree, $s(C_i)$ must be a descendant of $s(C_k)$ in T. □

We show that chains admit a tree structure.

Lemma 17. *The parent relation on C defines a tree U with $V(U) = C$ and root C_0.*

Proof. Let $D_0 \neq C_0$ be a chain in C and let $D_1, \dots, D_k$ be the sequence of chains containing the edges of $t(D_0) \to_T r$ in that order, omitting double occurrences. By definition of the parent relation, each D_i, $0 \leq i < k$, has parent D_{i+1}. It follows with $D_k = C_0$ that U is connected. Moreover, U is acyclic, as parent chains are always smaller in $<$ than their children due to the chain decomposition. □

For convenience, U will always denote the tree that is defined by the parent relation on C.

It remains to show how we can efficiently compute either a next BG-path for the current subgraph S_l, starting with $l = 3$, or a cut vertex or separation pair. For this purpose, we classify

the chains into different types in Section 5.3. We will eventually consider the chains $C_i \in C$ in the total order $<$ and focus on the chains that have a non-empty intersection with C_i in every step. Certain types of these chains will be BG-paths and therefore lead to the next subgraph in the construction sequence. The remaining ones will be grouped into bigger structures, called *caterpillars*, that can be decomposed into BG-paths later if the input graph is 3-connected (see Section 5.4).

To make the computation efficient, Section 5.5 restricts the desired construction sequence, which we try to compute, to a more special sequence. It is not clear that this restricted construction sequence for 3-connected input graphs exists; its existence is shown in Section 5.6. Section 5.7 deals with the computation of the restricted construction sequence in linear time by a reduction to interval overlaps.

5.3 Classification of Chains

We assign one of the Types 1, 2a, 2b, 3a and 3b to each chain $C_i \in C \setminus \{C_0\}$ in ascending order of $<$. Some of these types will be BG-paths under certain conditions. The types are determined by Algorithm 1 and dependent on the parent C_k of C_i: E. g., C_i is of Type 1 if $(t(C_i) \rightarrow_T s(C_i)) \subseteq C_k$ and of Type 2 if it is not of Type 1 and $s(C_i) = s(C_k)$. All chains are unmarked at the beginning of Algorithm 1. Note that chains that are backedges, in particular chains of Type 2a, cannot have children. We illustrate the different types in Figures 4, 5 and 12(b).

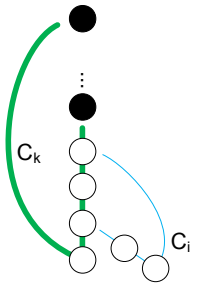
Algorithm 1 classify($C_i \in C \setminus \{C_0\}$, DFS-tree T) ▷ classifies chains into types 1,2a,2b,3a,3b

```
1:  Let C_k be the parent of C_i in U                                   ▷ C_k < C_i
2:  if t(C_i) →_T s(C_i) is contained in C_k then                       ▷ Type 1
3:      assign Type 1 to C_i
4:  else if s(C_i) = s(C_k) then          ▷ Type 2: C_k ≠ C_0, t(C_i) is inner vertex of C_k
5:      if C_i is a backedge then
6:          assign Type 2a to C_i                                       ▷ Type 2a
7:      else
8:          assign Type 2b to C_i; mark C_i                             ▷ Type 2b
9:  else                  ▷ Type 3: s(C_i) ≠ s(C_k), C_k ≠ C_0, t(C_i) is inner vertex of C_k
10:     if C_k is not marked then
11:         assign Type 3a to C_i                                       ▷ Type 3a
12:     else                                                            ▷ C_k is marked
13:         assign Type 3b to C_i; create a list L_i = {C_i}; C_j := C_k ▷ Type 3b
14:         while C_j is marked do                                      ▷ L_i is called a caterpillar
15:             unmark C_j; append C_j to L_i; C_j := parent(C_j)
```

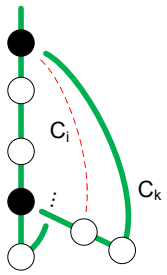
We first prove a basic property of chains of Types 2 and 3 and then show that the classification of chains can be carried out in linear time.

Lemma 18. *Let $C_i \neq C_0$ be a chain of Type 2 or 3 and let C_k be the parent of C_i. Then $C_k \neq C_0$ and $t(C_i)$ is an inner vertex of C_k.*

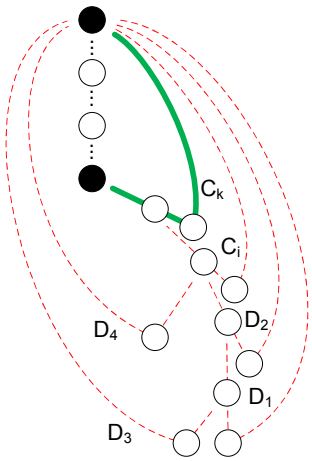
Proof. Assume to the contrary that $C_k = C_0$. Because $t(C_i)$ is contained in C_0, $s(C_i)$ must be in C_0 as well. But then C_i would be of Type 1, since $t(C_i) \rightarrow_T s(C_i) \subseteq C_0$. Therefore, $C_k \neq C_0$ holds



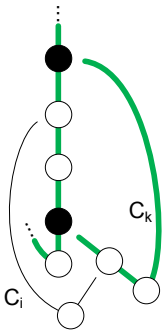
(a) C_i is of Type 1.



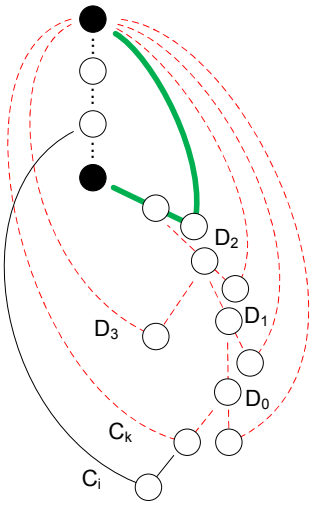
(b) C_i is of Type 2a, as it is a backedge.



(c) C_i and $D_1 - D_4$ are of Type 2b and therefore marked during the classification.



(d) C_i is of Type 3a.



(e) C_i is of Type 3b, as C_k is marked. The chain classification traverses the marked ancestors C_k, D_0, D_1 and D_2 of C_i, unmarks them and builds the list C_i, C_k, D_0, D_1, D_2 (the caterpillar L_i).

Figure 4: Different types of chains. The underlying DFS-tree is always depicted with straight edges, while backedges are bent; in addition, higher vertices are visited first in the DFS. Light solid (blue) chains are of Type 1, (red) dashed ones of Type 2 and black solid ones of Type 3.

and C_k must start with a backedge. Then the definition of the parent relation implies that $t(C_i)$ is an inner vertex of C_k. □

Lemma 19. *Classifying each chain with Algorithm 1 takes running time $O(m)$.*

Proof. In order to obtain a fast classification, we store the following information for each chain C_i: A pointer to its parent C_k (for $C_i \neq C_0$), pointers to $s(C_i)$ and $t(C_i)$ and the information whether C_i is a backedge. In addition, we store on each inner vertex of C_i a pointer to C_i. That allows us to check vertices on being contained as inner vertices or end vertices in arbitrary chains in $O(1)$. If $C_k = C_0$, C_i is of Type 1, as in that case $t(C_i)$ and $s(C_i)$ are contained in C_0 (see also Lemma 18). If $C_k \neq C_0$, C_i is of Type 1 if $s(C_i)$ is contained in $C_k \setminus s(C_k)$, which can be checked in constant time. The conditions for Type 2a and 2b need constant time as well. Every chain is marked at most once, therefore unmarked as most once in Line 15 of Algorithm 1, which gives a total running time of $O(m)$. □

Note that the classification with Algorithm 1 can be easily integrated into the chain decomposition. From now on, we assume that we have classified all chains.

We remark that the chains of Types 1 and 3 are identical to the paths that the *path-finding* procedure in the algorithm of Hopcroft and Tarjan [30] computes by using low-points, although the order is different. However, this does not hold for the chains of Types 2a and 2b.

We next define a necessary property for G to be 3-connected. Recall that $T(x)$ is the subtree of T that contains all descendants of x.

Property B: For every chain $C_i \in C \setminus \{C_0\}$ that is not a backedge and for its last inner vertex x, G contains a backedge e that enters $T(x)$ such that $s(e)$ is an inner vertex of $t(C_i) \rightarrow_T s(C_i)$.

We say that a chain C_i *has Property B* if C_i does not violate Property B. In particular, C_0 and every chain that is a backedge has Property B.

Lemma 20. *If a chain C_i violates Property B, $\{s(C_i), t(C_i)\}$ is a separation pair in G.*

Proof. The chain C_i can neither be C_0 nor a backedge. Let x be the last inner vertex of C_i. We first show that G contains a vertex y that is neither in $\{s(C_i), t(C_i)\}$ nor in $T(x)$. Assume otherwise. Then $s(C_i)$ must be the root of T, C_0 is the tree edge $s(C_i)t(C_i)$ in T and C_i is either C_1 or C_2, say C_1. As C_2 cannot contain inner vertices, C_2 must be the backedge $s(C_i)t(C_i)$, which contradicts the simpleness of G.

We show that every backedge that enters $T(x)$ must start at a vertex in $t(C_i) \rightarrow_T s(C_i)$. Clearly, such a backedge e starts at a proper ancestor of x due to the DFS-structure, i.e., it starts at an ancestor of $t(C_i)$. Moreover, e does not start at a proper ancestor of $s(C_i)$, as otherwise the chain containing e would have been traversed by the chain decomposition procedure before C_i and contain x. As Property B does not hold for C_i, we conclude that all backedges that enter $T(x)$ start either at $s(C_i)$ or $t(C_i)$. This causes $\{s(C_i), t(C_i)\}$ to be a separation pair in G, because its deletion separates the vertices y and x. □

Thus, Property B is necessary for G being 3-connected. The reader that is mainly interested in the computation of a construction sequence for a given 3-connected graph can therefore safely take Property B for granted. For the case that G is not known to be 3-connected, we show how to check Property B in linear time as part of the chain decomposition.

Lemma 21. *Property B can be checked in time $O(n+m)$ as part of the chain decomposition.*

Proof. We mark every chain that has Property B with a special marker during the chain decomposition. This gives a test on Property B in $O(n+m)$ time. Clearly, C_0 has Property B and we mark it in advance. Every chain that is a backedge has also Property B and we mark those chains as well (note that these are leaves in U, as they have no child). Whenever a chain $D_0 \neq C_0$ is found in the chain decomposition that is not of Type 2 (in fact, it suffices to deal with Type 3 chains), we traverse the path $P = D_0 \to_U C_0$ until a chain $C_j \neq D_0$ is reached that is already marked or contains $s(D_0)$. We mark every chain in $V(P) \setminus \{D_0, C_j\}$. The total running time for this procedure is $O(n+m)$, as no chain is marked twice.

It remains to show correctness. Let C_i be a chain that has Property B and assume to the contrary that C_i was not marked by the procedure. Then $C_i \neq C_0$ and C_i is not a backedge. Let x be the last inner vertex of C_i. Let e be the backedge that enters $T(x)$ with $s(e)$ being an inner vertex of $t(C_i) \to_T s(C_i)$ such that e was traversed first by the chain decomposition. Let D_0 be the chain that contains e and let $D_0, \dots, D_k, C_i$ be the vertices on the path $D_0 \to_U C_i$. As e is the first traversed backedge entering $T(x)$ such that $s(e)$ is an inner vertex of $t(C_i) \to_T s(C_i)$, Lemma 16 implies $s(D_1) = s(D_2) = \cdots = s(D_k) = s(C_i)$. In particular, D_0 cannot be of Type 2. It follows that D_0 was traversed by our procedure and that $s(D_0)$ is not contained in any of the chains $D_1, \dots, D_k, C_i$. As C_i is not marked by assumption, no chain in $D_0, \dots, D_k$ can be marked at this point in time in the chain decomposition. Thus, the procedure marks all chains $D_1, \dots, D_k, C_i$, which gives a contradiction.

Conversely, let C_i be a chain that does not have Property B. Then $C_i \neq C_0$ and C_i is not a backedge. Let x be the last inner vertex of C_i. Assume to the contrary that C_i was marked by the procedure and let e be the backedge that initiated the traversal that marked C_i. We denote the chain that contains e with D_0. With Lemma 16 and $C_i < D_0$, $s(e)$ must be a descendant of $s(C_i)$. As D_0 is by assumption not of Type 2 and $s(e)$ is not an inner vertex of $t(C_i) \to_T s(C_i)$, $s(e)$ must be a descendant of $t(C_i)$. But this contradicts that the traversal of e in the procedure marks C_i, as C_i contains $s(e)$. □

We check Property B by applying the algorithm of Lemma 21. If Property B is violated by a chain C_i, G has connectivity 2 due to Lemma 20 and the algorithm efficiently computes the separation pair $\{s(C_i), t(C_i)\}$. Otherwise, Property B is true; from now on, we will assume Property B.

We conclude this section with two direct consequences of Property B. Let a chain $C_i \neq C_0$ *enter* a subtree T' of a tree if the backedge in C_i enters T'. A chain in a subset of C is *minimal* if it is minimal with respect to $<$.

Lemma 22. *The chain C_0 contains an inner vertex.*

Proof. At least one of the chains C_1 and C_2, say C_1, is not a backedge, as G is simple and both chains have the same end vertices as C_0. Since C_1 has Property B, there is an inner vertex in $C_0 = t(C_1) \to_T s(C_1)$. □

Lemma 23. *Let $C_i \neq C_0$ be a chain that is not a backedge and let x be the last inner vertex in C_i. Then there is a chain C_j of Type 3 that enters $T(x)$ with $s(C_j)$ being an inner vertex of $t(C_i) \to_T s(C_i)$.*

Proof. According to Property B, there is a non-empty set X of backedges that enter $T(x)$ and start at an inner vertex of $t(C_i) \to_T s(C_i)$. Due to Lemma 13, every backedge in X is contained in exactly one chain. Let C_j be the minimal chain that contains a backedge in X. By definition of the types of the chains, C_j must be of Type 3. □

5.4 Caterpillars

Whenever a chain C_i of Type 3b is found in Algorithm 1, the path $C_i \to_U C_0$ is traversed until a chain C_j is found whose parent is not marked. The chains in $C_i \to_U C_j$ are stored in a list L_i and unmarked (see Line 15 of Algorithm 1 and Figure 4(e)). This way, every chain C_i of Type 3b is associated with a list L_i; we call each L_i a *caterpillar* (sometimes, we will regard L_i as a set instead of a list). Caterpillars group chains in order to handle them more easily. We give basic properties of caterpillars.

Lemma 24. *Every caterpillar L_i consists of exactly one chain of Type 3b, namely the chain C_i, and one or more chains of Type 2b.*

Proof. Clearly, $C_i \in L_i$ (see Line 13 in Algorithm 1). The claim follows directly from the fact that only chains of Type 2b are marked and the definition of Type 3b. □

Lemma 25. *The set of chains $C \setminus \{C_0\}$ is partitioned into the chains of Types 1, 2a and 3a and the chains being contained in caterpillars. Moreover, no chain is contained in two caterpillars.*

Proof. With Lemma 24, it remains to show that every chain C_i of Type 2b is contained in exactly one caterpillar. At the time C_i was classified as Type 2b, C_i was marked. Any chain of Type 2b can only be added to a caterpillar if it is marked. If such a marked chain is added to a caterpillar, it immediately becomes unmarked for the rest of Algorithm 1 (see Line 15 of Algorithm 1). We conclude that C_i can be in at most one caterpillar; to show that C_i is contained in exactly one caterpillar, we prove that Algorithm 1 unmarks C_i.

Let C_k be the parent of C_i. Because C_i is of Type 2b, $C_i \neq C_0$ and C_i is not a backedge. Let x be the last inner vertex of C_i. According to Lemma 23, there is a chain of Type 3 that enters $T(x)$ and starts at an inner vertex in $t(C_i) \to_T s(C_i)$. Let C_j be the minimal such chain. Immediately before C_j is found in the chain decomposition, every chain that ends at a vertex in $T(x)$ must start at $s(C_i)$ with Lemma 16 and is therefore of Type 2a or 2b. Since chains of Type 2a are backedges and cannot have children, the parent of C_j must be of Type 2b and is therefore marked. Moreover, every chain corresponding to a vertex in $V(C_j \to_U C_i) \setminus \{C_j\}$ is of Type 2b and marked. Thus, C_j is of Type 3b and Algorithm 1 unmarks C_i (see Line 15 of Algorithm 1). □

The above arguments show also that the chain of Type 3b in a caterpillar cannot start at an arbitrary vertex. We get the following result.

Lemma 26. *Let L_i be a caterpillar and D_k be the minimal chain in L_i. Then $s(C_i)$ is an inner vertex of $t(D_k) \to_T s(D_k)$.*

Proof. Let x be the last inner vertex of D_k. According to the construction of caterpillars, C_i is the minimal chain that ends on a vertex in $T(x)$ and is neither of Type 1 nor Type 2. With Lemma 23, $s(C_i)$ is an inner vertex in $t(D_k) \to_T s(D_k)$. □

Definition 27. Let the *parent of a caterpillar* L_i be the parent of the minimal chain in L_i.

For computing the next BG-paths, we will often consider caterpillars as whole or chains that are not contained in caterpillars.

Definition 28. A *cluster* is either a caterpillar or a chain of Type 1, 2a or 3a.

With Lemma 25, every chain $C_i \neq C_0$ is contained in exactly one cluster. The *cluster of a chain* is the cluster that contains the chain. The *clusters of a set* X of chains are the clusters of chains in X, omitting double occurrences. We extend the strict total order $<$ on chains to clusters.

Definition 29. For two clusters A and B, let $A < B$ if there is a chain C_a in A and a chain C_b in B with $C_a < C_b$.

Note that the relation $<$ on clusters is still a strict total order, as caterpillars correspond to vertex-disjoint paths P in U with the property that one end vertex of P is a proper ancestor of the other end vertex in U. Recall that we defined a pre-order only for the vertices of a forest, e. g., $<$ is a pre-order on $V(U)$. For completeness, we extend pre-orders to clusters. Note that the ancestors of cluster are well-defined by the given parent-relations for chains and caterpillars.

Definition 30. Let a strict total order $\prec$ on a set of clusters F be a *pre-order* if, for every cluster $A \in F$, all proper ancestors of A in F precede A in $\prec$.

5.5 Restrictions

We impose restrictions on the construction sequence that will simplify the computation. Recall that $S_3, \ldots, S_{m-n+2}$ is the sequence of generated subgraphs of G in the construction sequence when G is 3-connected.

Definition 31. Let S_l, $3 \leq l \leq m - n + 2$, be *upwards-closed* if, for each vertex v in S_l, the edge from v to its parent in T is contained in S_l. Let S_l be *modular* if S_l is the union of chains.

Thus, if a modular and upwards-closed subgraph S_l contains a chain C_i, S_l must contain also the parent of C_i. Clearly, S_3 is upwards-closed and modular. We would be done if we could restrict every S_l to be upwards-closed and modular, as then every chain would be a BG-path. However, this is not possible, as the following example shows. Consider $S_3 = \{C_0, C_1, C_2\}$ in the graph of Figure 5. As every BG-path for S_3 has end vertices x and y, S_4 cannot be modular.

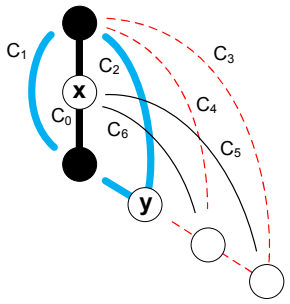
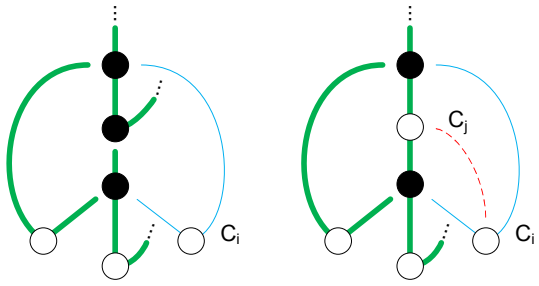


Figure 5: C_1 and C_2 are of Type 1, C_3 is of Type 2b, C_4 of Type 2a, C_5 of Type 3b and C_6 of Type 3a. No BG-path for the subgraph S_3 (depicted with thick blue edges) preserves modularity.

Therefore, we impose the following weaker restriction (R_1): We add only clusters that can be decomposed into subsequent BG-paths and whose additions generate upwards-closed and modular

subgraphs. We additionally demand that each cluster is decomposed into as many BG-paths as it contains chains. Thus, if the cluster is not a caterpillar, just a chain of Type 1, 2a or 3a is added that is a BG-path. Note that intermediate BG-paths of clusters that are caterpillars may violate upwards-closedness and modularity.



(a) allowed (b) forbidden

Figure 6: The effect of Restriction (R_2) on adding the BG-path C_i to the fat subgraph. If (R_2) would not forbid to add C_i in Figure (b), adding chain C_j next could violate Property 4.3.

For the generated subgraph S_{l+t}, we impose also the restriction (R_2) that no link in S_{l+t} that consists only of tree edges has a parallel link. This restriction will cause certain BG-path candidates (e.g., certain chains) to automatically satisfy Property 4.3 of the BG-path definition 4 due to the DFS-structure (see Figure 6). It implies also that the BG-path that is applied on S_3 generates a K_4-subdivision and not a K_2^4-subdivision. This is necessary for a sequence (7) by definition. Note that (R_2) does not hold for S_3, as C_0 has the parallel links C_1 and C_2. It must however hold for all following subgraphs. We summarize the restrictions.

Restrictions: Let $S_l \subset G$ be the current upwards-closed and modular subgraph. We will only add a cluster that

- (R_1) – can be decomposed into as many subsequent BG-paths as it contains chains and that
  – creates an upwards-closed and modular subgraph S_{l+t} such that
- (R_2) – no link in S_{l+t} that consists only of tree edges has a parallel link in S_{l+t} (note that $S_{l+t} \neq S_3$).

In particular, Restriction (R_1) forces the total number of operations in the construction sequence to be $|C \setminus \{C_0, C_1, C_2\}| = |C| - 3$. According to Lemma 13, $|C| - 3 = m - n - 1$, which is necessary for every sequence (7), as shown in Lemma 10.

From now on, we will only deal with construction sequences that are restricted by (R_1) and (R_2). Whenever we are searching for new BG-paths, the current subgraph S_l is upwards-closed, modular and consists of exactly l chains. We denote S_l by S_l^R in such cases to emphasize these properties. It is not clear whether such a restricted construction sequence exists; we will prove its existence in Section 5.6.

Let a cluster that satisfies (R_1) and (R_2) on S_l^R be *addable*. We first show that Restriction (R_2) implies Property 4.3.

Lemma 32. *Every path P for S_l^R with Properties 4.1 and 4.2 is a BG-path. If P is additionally a chain of Type 2a or 3a, P is addable.*

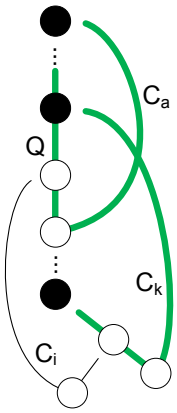


Figure 7: The chain C_i of Type 3 is addable to the fat subgraph.

Proof. For the first claim, assume to the contrary that P violates Property 4.3. Then $|V_{real}(S_l^R)| \geq 4$ must hold and $S_l^R \neq S_3^R$ follows. Let Q and Z be the parallel links of S_l^R that contain the end vertices of P as inner vertices, respectively. Both links, Q and Z, must contain a backedge, as otherwise one of them would contain only tree edges and (R_2) would be violated, since $S_l^R \neq S_3^R$.

Let $C_i \neq C_0$ be the chain in S_l^R that contains Q and let $C_j \neq C_0$ be the chain in S_l^R that contains Z. According to Proposition 14, C_i and C_j contain each exactly one backedge (the first edge). This implies that $s(C_i)$ is an end vertex of Q, $s(C_j)$ is an end vertex of Z and $s(C_i) = s(C_j)$. Let v be the vertex in $Q \cap Z$ that is different from $s(C_i)$. By construction of the chain decomposition, the inner vertices of Q and Z are contained in disjoint subtrees of T. Since T is a DFS-tree, P must contain an inner vertex that is an ancestor of v. As S_l^R is upwards-closed, this vertex is already contained in S_l^R, which contradicts P to have Property 4.1.

For the second claim, let P be a chain of Type 2a or 3a. Since P is a chain, S_{l+1}^R is upwards-closed and modular and (R_1) is satisfied. By definition of Types 2 and 3, $t(P) \to_T s(P)$ is not contained in a chain in S_l^R and therefore has an inner vertex that is the end vertex of a chain in S_l^R. As this vertex is real, adding P preserves (R_2) if $S_l^R \neq S_3^R$. In the remaining case $S_l^R = S_3^R$, P must be of Type 3a, as otherwise P would contradict Property 4.2. Then adding P must induce an inner real vertex in C_0, which satisfies (R_2) for S_4^R. □

We show under which conditions chains of Types 1, 2a and 3a are addable to S_l^R if not already contained in S_l^R.

Lemma 33. *Let $C_i \neq C_0$ be a chain such that the parent C_k of C_i but not C_i itself is contained in S_l^R. If C_i is either of Type 1 with an inner real vertex in $t(C_i) \to_T s(C_i)$, of Type 2a with a real vertex in $(t(C_i) \to_{C_k} s(C_i)) \setminus s(C_i)$ or of Type 3a, C_i is addable.*

Proof. Since S_l^R is upwards-closed, modular and contains C_k, C_i satisfies Property 4.1 in all cases. Let C_i be of Type 1. Then the inner real vertex in $t(C_i) \to_T s(C_i)$ prevents any link that contains both, $s(C_i)$ and $t(C_i)$, from having $s(C_i)$ or $t(C_i)$ as an inner vertex. This causes C_i to have Property 4.2. Lemma 32 implies that C_i is a BG-path for S_l^R. As adding C_i preserves S_{l+1}^R to be upwards-closed and modular, (R_1) is satisfied. Due to the inner real vertex in $t(C_i) \to_T s(C_i)$, S_l^R must be different from S_3^R and (R_2) holds in S_{l+1}^R. It follows that C_i is addable.

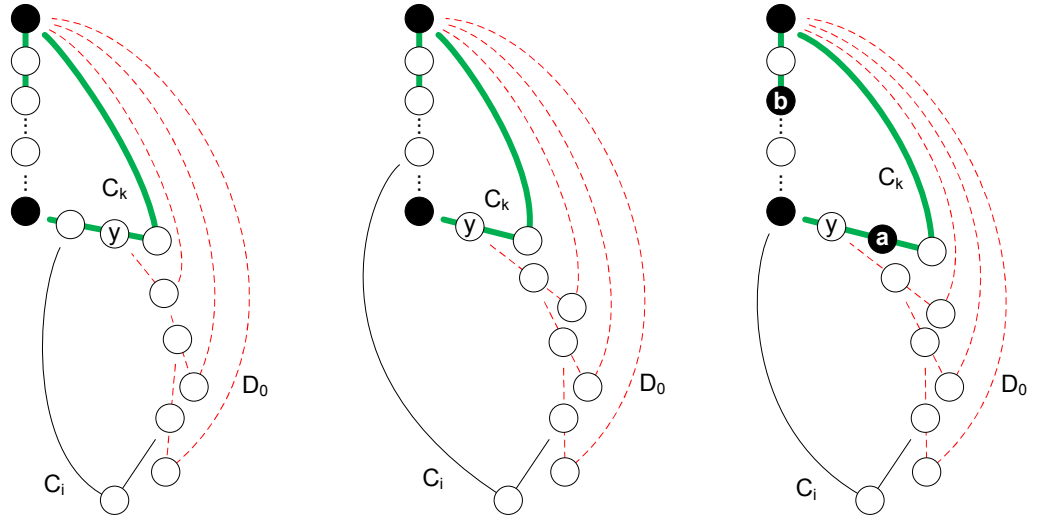
Let C_i be of Type 2a. The vertex $s(C_i)$ is real, as it is the end vertex of a chain in S_l^R. If additionally $t(C_i)$ is real, every link in S_l^R that contains $s(C_i)$ and $t(C_i)$ must contain $s(C_i)$ and

$t(C_i)$ as end vertices. Otherwise, $t(C_i) \to_{C_k} s(C_i)$ contains an inner real vertex by assumption and $t(C_i)$ must be an inner vertex of a link in S_l^R that does not contain $s(C_i)$, as $t(C_k)$ is real. Both cases ensure that C_i has Property 4.2. By Lemma 32, C_i is addable.

Let C_i be of Type 3a. Then $s(C_i) \neq s(C_k)$ holds by definition and $C_k \neq C_0$, as otherwise C_i would be of Type 1. Additionally, $C_k < C_i$, since C_k is the parent of C_i. According to Lemma 16, $s(C_i)$ must be an inner vertex of the path $t(C_k) \to_T s(C_k)$ (see Figure 7). Therefore, every chain C_j that contains both, $s(C_i)$ and $t(C_i)$, satisfies $C_i \cap C_j = \{s(C_i), t(C_i)\} = \{s(C_j), t(C_j)\}$. This causes C_i to have Property 4.2. By Lemma 32, C_i is addable. □

According to Lemma 33, the condition for adding a chain $C_i \not\subseteq S_l^R$ of Type 3a is very simple: It suffices that its parent is contained in S_l^R. This gives a valuable algorithmic approach: Whenever a chain in S_l^R has children of Type 3a in U that are not already in S_l^R, these children are addable. We next give similar conditions for the remaining Types 2b and 3b. According to Lemma 25, these chains are exactly the ones that are contained in caterpillars.

Definition 34. Let a caterpillar L_i with parent C_k be *bad* for S_l^R if $s(C_i)$ is contained in C_k and $s(C_i) \to_{C_k} s(C_k)$ contains no inner real vertex (see Figure 8(a)). Otherwise, L_i is called a *good* caterpillar (see Figures 8(b) and (c)).



(a) A *bad* caterpillar L_i with parent C_k.

(b) A *good* caterpillar L_i with parent C_k such that $s(C_i)$ is an inner vertex of $t(C_k) \to_T s(C_k)$.

(c) A *good* caterpillar L_i with parent C_k, $s(C_i) \in V(C_k)$ and an inner real vertex a in $s(C_i) \to_{C_k} s(C_k)$.

Figure 8: Kinds of caterpillars.

Let L_i be a bad caterpillar with parent C_k and let y be the last vertex of the minimal chain in L_i. According to Lemma 26, $s(C_i) \in V(t(C_k) \to_{C_k} y) \setminus \{y\}$. We characterize good caterpillars.

Lemma 35. *A caterpillar L_i with parent C_k is good if $s(C_i)$ is either an inner vertex of $t(C_k) \to_T s(C_k)$ (see Figure 8(b)) or a vertex in C_k such that $s(C_i) \to_{C_k} s(C_k)$ contains an inner real vertex (see Figure 8(c)).*

Proof. If $s(C_i) \notin V(C_k)$, $s(C_i)$ must be an inner vertex of $t(C_k) \to_T s(C_k)$ with Lemma 26. Otherwise, $s(C_i) \in V(C_k)$ and the path $s(C_i) \to_{C_k} s(C_k)$ contains an inner real vertex, as otherwise L_i would be bad. □

We show that good caterpillars are addable under minor assumptions.

Lemma 36. *Let L_i be a caterpillar such that the parent C_k of L_i but no chain in L_i is contained in S_l^R. If L_i is good, L_i is addable.*

Proof. Let L_i be good and let $t > 1$ be the number of chains in L_i. Clearly, the graph generated by adding all chains in L_i to S_l^R is upwards-closed and modular, as L_i consists of consecutive ancestors of C_i in U. It remains to show that L_i can be decomposed into t successive BG-paths for S_l^R that generate the subgraphs $S_{l+1}, S_{l+2}, \dots, S_{l+t}$ such that S_{l+t} satisfies (R_2). Let y be the last vertex of the minimal chain in L_i, thus $y \in V(C_k)$.

We assume at first that $s(C_i)$ is an inner vertex of $t(C_k) \to_t s(C_k)$ (see Figure 8(b)). Then the path $P = C_i \cup (t(C_i) \to_T y)$ fulfills Properties 4.1 and 4.2 and is a BG-path for S_l^R with Lemma 32. Note that adding P preserves S_{l+1} to be upwards-closed but not modular. Successively, for each chain C_j of the $t-1$ chains in $L_i \setminus \{C_i\}$ (in arbitrary order), we add the path $s(C_j) \to_{C_j} v$ with v being the first vertex in C_j that is in P. All these paths are BG-paths, because y is real in S_{l+1}.

Now assume with Lemma 35 that $s(C_i)$ is contained in C_k with an inner real vertex a in $s(C_i) \to_{C_k} s(C_k)$ (see Figure 8(c)). We first show that $t(C_k) \to_T s(C_k)$ contains an inner real vertex as well. Assume the contrary. Then C_k must be of Type 1 and contradicts (R_2), unless $S_l^R = S_3^R$. But S_l^R must be different from S_3^R, since a exists, and it follows that $t(C_k) \to_T s(C_k)$ contains an inner real vertex b.

Let D_0 be the parent of C_i, which must be contained in L_i, as $t > 1$. Then $(C_i \cup D_0) \setminus ((t(C_i) \to_T y) \setminus t(C_i))$ is a BG-path due to the real vertices a and b and we add it, although it neither preserves S_{l+1} to be upwards-closed nor modular. We next add $t(C_i) \to_T y$, which restores upwards-closedness. Successively, for each chain C_j of the $t-2$ remaining chains in $L_i \setminus \{C_i, D_0\}$ (in arbitrary order), we add the path $s(C_j) \to_{C_j} v$ with v being the first vertex in C_j that is contained in $V(t(C_i) \to_T y)$. With the same line of argument as before and Lemma 32, all these paths are BG-paths.

We show that S_{l+t} satisfies (R_2). Let Q be a link in S_{l+t} that consists only of tree edges and that is not a link in S_l^R, i. e., Q is generated when adding L_i. If Q is contained in L_i, Q has no parallel link in S_{l+t} by construction. Otherwise, Q must be contained in a link in S_l^R that was subdivided by at least one of the real vertices $s(C_i)$ and y when adding L_i. In this case, Q has no parallel link in S_{l+t}, as $t(C_i)$ is real in S_{l+t} for both decompositions of L_i.

Otherwise, let Q be a link in S_{l+t} that consists only of tree edges and is a link in S_l^R as well. Then $S_l^R \neq S_3^R$ holds, as otherwise $Q = C_0$ and adding L_i would induce an inner real vertex in Q, contradicting the choice of Q. It follows with (R_2) that Q has no parallel link in S_l^R. Then Q cannot have a parallel link in S_{l+t}, as every path between two of the three vertices $\{s(C_i), y, s(C_k)\}$ in the union of chains in L_i contains an inner real vertex in S_{l+t}. We conclude that S_{l+t} satisfies (R_2) and that L_i is addable. □

Note that the decomposition of a good caterpillar L_i into subsequent BG-paths as shown in Lemma 36 can be computed in time linearly dependent on the edges in L_i.

5.6 Existence of the Restricted Sequence

We show that, even under the Restrictions (R_1) and (R_2), a construction sequence from S_3^R to G exists.

Definition 37. Let $\sim$ be the equivalence relation on $E(G) \setminus E(S_l)$ such that

- $\forall e, f \in E(G) \setminus E(S_l) : e \sim f$ if $e = f$ or there is a path in G that contains e and f but no vertex of S_l as inner vertex.

Let a subgraph H of a graph G be *edge-induced* by an edge set $E' \subseteq E(G)$ if $E(H) = E'$ and $V(H)$ is the union of the end vertices of all edges in E'.

Definition 38. Let the *segments* of S_l be the subgraphs of G that are edge-induced by the equivalence classes of $\sim$. For a segment H of S_l, let $V(H) \cap V(S_l)$ be the *attachment vertices* of H.

It is important to note that every segment of S_l^R is the union of all vertices in a subtree of U (we say of all *chains* in this subtree), as S_l^R is modular and upwards-closed. For a chain C_i that is not contained in S_l, let the *segment* of C_i be the segment of S_l that contains C_i. Sometimes, we will identify a segment with the set of the chains it contains.

The concept of segments is not new. Starting with the work of Auslander and Parter [2], segments were used to design many efficient algorithms for problems related to planarity [26, 12, 31, 41, 37, 49, 68, 71] and 3-connectivity [30, 67, 68]. It is folklore that a segment of a cycle in a 3-connected graph is either an edge or has at least three attachment vertices. Due to Lemma 23, we can deduce this result (in our notation) also for G, although G is not known to be 3-connected.

Lemma 39. *Every segment H of S_l^R that is not a backedge has at least three attachment vertices.*

Proof. Let C_i be the minimal chain in H. Since $t(C_i) \in S_l^R$ and S_l^R is upwards-closed, both vertices $s(C_i)$ and $t(C_i)$ are attachment vertices of H. The chain C_i is not a backedge, as otherwise H would be a backedge. According to Lemma 23, H contains a chain that starts at an inner vertex x of $t(C_i) \rightarrow_T s(C_i)$. As S_l^R is upwards-closed, x is a third attachment vertex of H. □

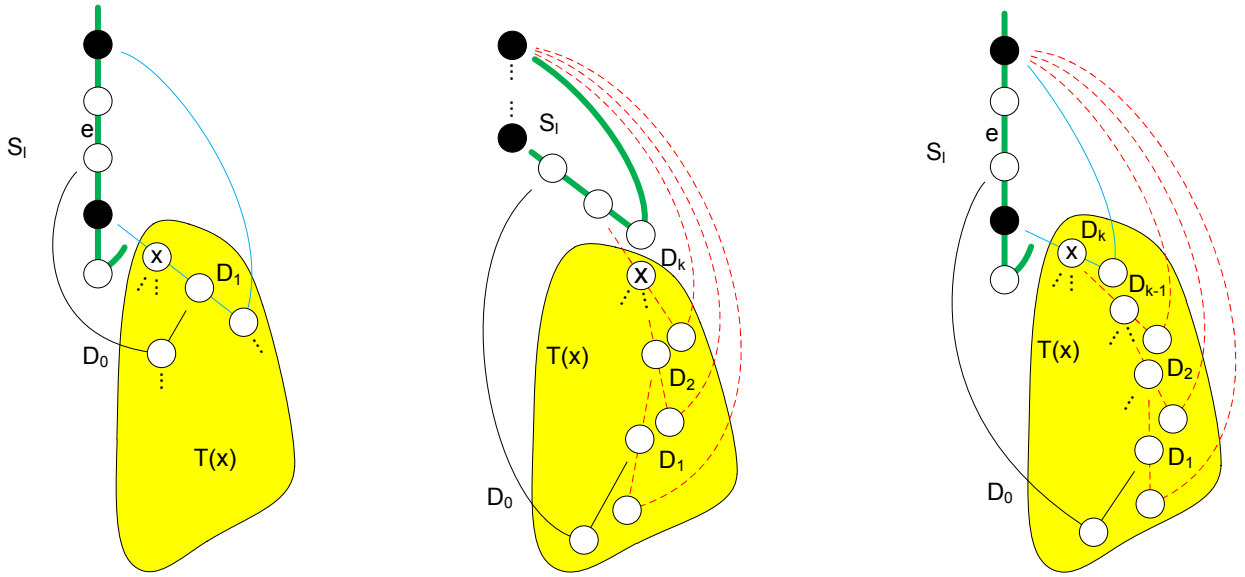
A chain whose parent is in S_l^R but which is not contained in S_l^R itself is of interest, as it is a possible candidate for a BG-path.

Lemma 40. *Every segment H of S_l^R contains exactly one chain that is a child of a chain in S_l^R, namely the chain that is minimal in H.*

Proof. Recall that upwards-closedness and modularity of S_l^R implies that there is a subtree U' of U such that H is the union of the chains in U'. Clearly, only the root D_k of U' (i. e., the minimal chain in H) can be a child of a chain in S_l^R. The chain D_k is a child of a chain in S_l^R, as $t(D_k) \in S_l^R$, S_l^R is upwards-closed and S_l^R contains at least the root C_0 of U. □

We show in which cases chains of Type 3 that start in S_l^R but are not contained in S_l^R are addable.

Lemma 41. *Let D_0 be a chain of Type 3 such that $s(D_0) \in V(S_l^R)$, $D_0 \not\subseteq S_l^R$ and D_0 is minimal among the chains of Type 3 in its segment H. Let $D_k < \cdots < D_0$ be all ancestors of D_0 that are contained in H. Then the clusters of $D_k, \ldots, D_0$ are successively addable to S_l^R, unless one of the following exceptions holds.*



(a) Exception 41.1 (b) Exception 41.2 (c) Exception 41.3

Figure 9: The three exceptions of Lemma 41. The black vertices in Exceptions 41.1 and 41.3 may also be non-real.

1. *D_0 is of Type 3a, $k = 1$, D_k is of Type 1, $s(D_0)$ is an inner vertex of $t(D_k) \to_T s(D_k)$ and there is no inner real vertex in $t(D_k) \to_T s(D_k)$ (see Figure 9(a)),*
2. *D_0 is of Type 3b and $\{D_0, \dots, D_k\}$ is a bad caterpillar (see Figure 9(b)),*
3. *D_0 is of Type 3b, $\{D_0, \dots, D_{k-1}\}$ is a caterpillar, D_k is of Type 1, $s(D_0)$ is an inner vertex of $t(D_k) \to_T s(D_k)$ and there is no inner real vertex in $t(D_k) \to_T s(D_k)$ (see Figure 9(c)).*

Proof. Due to the choice of D_0, there is no chain of Type 3 in $\{D_1, \dots, D_k\}$. The chain D_k is minimal in H. Additionally, $\{D_1, \dots, D_k\}$ does not contain a chain of Type 2a, as chains of that type cannot have children.

Let D_0 be of Type 3a. If $k = 0$, $t(D_0)$ is contained in S_l^R and D_0 is addable with Lemma 33, implying the claim. Otherwise, $k \geq 1$. Assume that D_1 is of Type 2b and let $C_j \neq D_0$ be the chain of Type 3b in the caterpillar containing D_1. Then C_j is contained in H, as S_l^R is upwards-closed and modular. Moreover, $C_j < D_0$ holds, as otherwise D_0 would have been of Type 3b in the chain decomposition. This contradicts the minimality of D_0 and it only remains that D_1 is of Type 1.

The vertex $s(D_1)$ is a proper ancestor of $s(D_0)$, since $D_1 < D_0$ and D_0 is not of Type 2. Since D_0 is not of Type 1, $s(D_0)$ must be a proper ancestor of $t(D_1)$. It follows that $s(D_0)$ is an inner vertex of $t(D_1) \to_T s(D_1)$. If $k \geq 2$, D_2 must contain $t(D_1) \to_T s(D_1)$, because D_1 is of Type 1 and a child of D_2. Hence, the edge e joining $s(D_0)$ with the parent of $s(D_0)$ in T is contained in D_2 (see Figure 9(a)). But since S_l^R is upwards-closed and $s(D_0) \in V(S_l^R)$, e must be contained in S_l^R, contradicting that $k \geq 2$. Thus, $k = 1$ and the parent of D_1 in U is contained in S_l^R. If $t(D_1) \to_T s(D_1)$ contains an inner real vertex, D_1 and D_0 are subsequently addable with Lemma 33. Otherwise, Exception 41.1 holds.

Let D_0 be of Type 3b and let L_i be the caterpillar that contains D_0. Due to (R_1), every chain in L_i is contained in H and, by definition of caterpillars, D_1 is of Type 2b and in L_i. Let D_t,

$1 \leq t \leq k$, be the minimal chain in L_i. If $t = k$ and L_i is good, the parent of L_i is contained in S_l^R and L_i is addable with Lemma 36. If $t = k$ and L_i is bad, Exception 41.2 holds (see Figure 9(b)).

The only remaining case is $t < k$. Assume that D_{t+1} is of Type 2b and let C_j be the chain of Type 3b in the caterpillar containing D_{t+1}. Then, C_j is contained in H and $C_j < D_0$ holds. This contradicts the minimality of D_0 and we conclude that D_{t+1} is of Type 1 (see Figure 9(c)).

As $D_{t+1} < D_0$ and D_0 is not of Type 2, $s(D_{t+1})$ is a proper ancestor of $s(D_0)$. Since D_{t+1} has a child, D_{t+1} is not a backedge. Applying Lemma 23 to the chain D_{t+1} implies together with the minimality of D_0 that $s(D_0)$ is an inner vertex of $t(D_{t+1}) \to_T s(D_{t+1})$. The parent of D_{t+1} is in S_l^R, as it contains $t(D_{t+1}) \to_T s(D_{t+1})$ and $s(D_0) \to_T s(D_{t+1})$ is contained in S_l^R. This implies $k = t + 1$. If there is no inner real vertex in $t(D_k) \to_T s(D_k)$, Exception 41.3 holds. Otherwise, Lemma 33 implies that D_k is addable. After adding D_k, L_i is good due to Lemma 35 and addable with Lemma 36. □

We extend Lemma 41 to all chains of Type 3 that start at a vertex in S_l^R. Let a caterpillar L_i *start* at the vertex $s(C_i)$.

Lemma 42. *Let the preconditions of Lemma 41 hold and let D_0 be not contained in one of the Exceptions 41.1–41.3. Let Y be the set of ancestors of all chains in H that are of Type 3 and start in S_l^R. Then the clusters of Y are successively addable in any pre-order that adds clusters that start at $t(D_k)$ last, e. g., in ascending order of $<$.*

Proof. We show how the clusters of Y are successively addable by iteratively applying Lemma 41. Note that Y is the vertex set of a subtree U_Y of U that is rooted on D_k (D_k is defined by the preconditions of Lemma 41). By definition of Y, the leaves of U_Y are chains of Type 3 that start at a vertex in S_l^R.

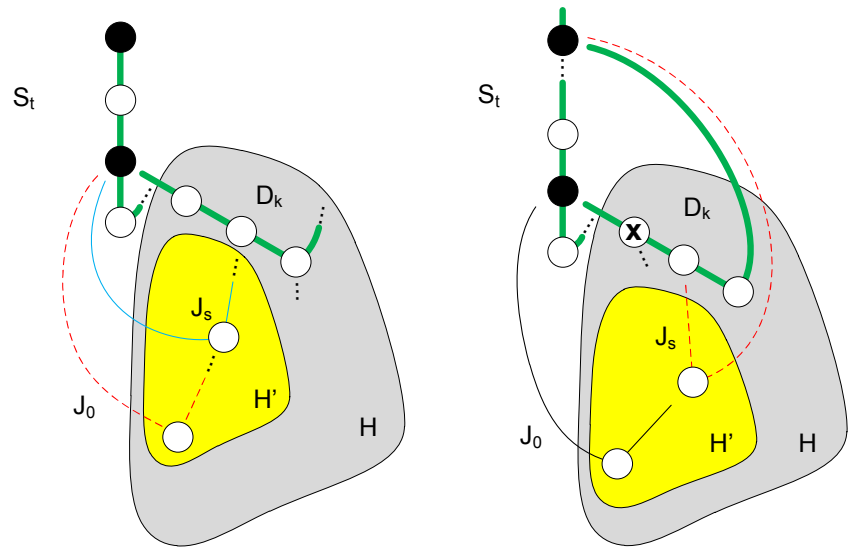
Let any pre-order σ on the clusters of Y be given that adds clusters that start at $t(D_k) \in V(S_l^R)$ last. We go along σ. After adding an arbitrary number of clusters in the order of σ, let J be the next cluster to add. Let J_s be the minimal chain in J and let S_t^R be the current subgraph of G. At this point, J_s is the root of a subtree U' of U_Y and, as σ is a pre-order, the minimal chain in its segment H' of S_t^R. Let J_0 be the minimal chain of Type 3 in $V(U')$ that starts at a vertex in S_l^R. Then J_0 is also the minimal such chain in H'. Let $J_s < \cdots < J_0$ be all ancestors of J_0 in U'.

We apply Lemma 41 to J_0 in H' and show that J_0 cannot be contained in one of the Exceptions 41.1–41.3. This implies that the clusters of $J_s, \ldots, J_0$ are addable in pre-order. The claim follows from merely adding J and iterating the argument for the next cluster in σ.

It remains to show that J_0 is not contained in one of the Exceptions 41.1–41.3. By assumption, this holds for $J_0 = D_0$; in that case, the cluster of D_k is added. As σ is a pre-order, the cluster of D_k is the first cluster in H that is added. We can therefore assume for the following arguments that D_k is in S_t^R.

First, assume to the contrary that J_0 is contained in Exception 41.1 or 41.3 (see Figure 10(a)). Since S_l^R is upwards-closed, $s(J_0) \in V(S_l^R)$ implies that $s(J_s) \in V(S_l^R)$. Because of Lemma 40, D_k is the only chain in H that ends on a vertex in S_l^R. Since J_s is a proper descendant of D_k and of Type 1, it follows from $s(J_s) \in V(S_l^R)$ that $s(J_s) = t(D_k)$. As $J_0 > J_s$ and $s(J_0) \in V(S_l^R)$, $s(J_0) = s(J_s) = t(D_k)$ holds due to Lemma 16. This contradicts J_0 to be in Exception 41.1 or 41.3, because $s(J_0)$ is not an inner vertex of $t(J_s) \to_T s(J_s)$.

Now assume to the contrary that J_0 is contained in Exception 41.2 (see Figure 10(b)). Then J_0 is of Type 3b and part of a bad caterpillar L_j in S_t^R, whose parent D is not contained in H'. We show that $D = D_k$. Because L_j contains only chains in $H' \subset H$ and D_k is the minimal chain in



(a) J_0 is not contained in Exceptions 41.1 and 41.3.

(b) J_0 is not contained in Exception 41.2.

Figure 10: In S_t^R, J_0 is not contained in one of the Exceptions 41.1–41.3.

H that is already contained in S_t^R, D must be a descendant of D_k. Since L_j is bad in S_t^R, $s(J_0)$ is contained in $D \setminus s(D)$. By definition of J_0, $s(J_0) \in V(S_l^R)$. Because of Lemma 40, D_k is the only chain in H that ends on a vertex in S_l^R. Moreover, no inner vertex of a chain in H is contained in S_l^R, as H does not contain C_0 and S_l^R is upwards-closed. It follows that $D = D_k$ and $s(J_0) = t(D_k)$.

We show that L_j cannot be bad in S_t^R. The chain D_k is not a backedge, as it has the child J_s. Let x be the next to last vertex of D_k. Applying Lemma 23 on D_k implies that a chain C_y of Type 3 enters $T(x)$ with $s(C_y)$ being an inner vertex of $t(D_k) \to_T s(D_k)$. The chain C_y is contained in H, but not in H', as that would contradict the choice of J_0. As the pre-order σ adds clusters that do start at $t(D_k)$ last, the clusters of C_y and of all proper ancestors of C_y in H must have been added before. Thus, D_k must contain at least one inner real vertex in S_t^R, which contradicts L_j to be bad. □

Definition 43. For S_l^R and a chain C_i in S_l^R, let $\mathit{Children}_{12}(C_i)$ be the set of children of C_i of Types 1 and 2 that are not contained in S_l^R and let $\mathit{Type}_3(C_i)$ be the set of chains of Type 3 that start at a vertex in C_i and are not contained in S_l^R.

The definition of $\mathit{Children}_{12}(C_i)$ and $\mathit{Type}_3(C_i)$ for a chain C_i in S_l^R is crucial, as the clusters of these sets are essentially the ones for which we will prove that they can are addable under a certain condition. We defined $\mathit{Children}_{12}(C_i)$ to contain only children of Type 1 and 2. On a high-level view, it will not be necessary to consider children of C_i of Type 3, as these children (and their clusters) are already contained in S_l, as they have been added when the clusters of $\mathit{Type}_3(C_j)$ for an ancestor C_j of C_i were considered.

We start by showing that, under a certain condition, the clusters of the following subset of $\mathit{Type}_3(C_i)$ are addable.

Lemma 44. *Let C_i be a chain in S_l^R such that $\text{Type}_3(C_j) = \emptyset$ holds for every proper ancestor C_j of C_i. Let B be the subset of chains in $\text{Type}_3(C_i)$ whose segments do not contain a chain in $\text{Children}_{12}(C_i)$. Then the clusters of all ancestors of chains in B that are not in S_l^R are successively addable. The order of addition can be any pre-order that adds clusters in same segments of S_l^R consecutively and in which the start vertices of the clusters of B are consecutive in $t(C_i) \rightarrow_{C_i} s(C_i)$, e. g., in ascending order of $<$.*

Proof. Let C_y be a chain in B such that $s(C_y)$ is an ancestor of $s(C_z)$ for every chain $C_z \in B$. Let H be the segment of C_y. We show that the clusters of all ancestors of chains in $B \cap H$ are addable. Then choosing C_y as before for the remaining subset of B and iterating the argument on C_y gives the claim.

Let D_0 be the minimal chain of Type 3 in H. We show that $D_0 \in B$. According to Lemma 16, $s(D_0)$ is an ancestor of $s(C_y)$. However, $s(D_0)$ cannot be contained in a proper ancestor C_j of C_i, as $\mathit{Type}_3(C_j) = \emptyset$ holds by assumption. It follows that $s(D_0)$ starts at a vertex in C_i and, thus, $D_0 \in B$.

We want to apply Lemma 42 on D_0 to add the clusters of all ancestors of chains in $B \cap H$. Let D_k be the minimal chain in H. To apply Lemma 42, it suffices to show that D_0 is not contained in one of the Exceptions 41.1–41.3. Note that this does not directly follow from the fact that $D_k \notin \mathit{Children}_{12}(C_i)$, as D_k does not need to be a child of C_i for Exceptions 41.1–41.3.

First, assume to the contrary that D_0 is contained in Exception 41.1 or 41.3. Then D_k is of Type 1. It suffices to show that C_i is the parent of D_k, as this would contradict the assumption that $H \cap \mathit{Children}_{12}(C_i) = \emptyset$. Since D_k is of Type 1, the path $t(D_k) \rightarrow_T s(D_k)$ is contained in the parent of D_k. In both Exceptions, $s(D_0)$ is an inner vertex of $t(D_k) \rightarrow_T s(D_k)$ and, in addition, $s(D_0)$ is not real, as $t(D_k) \rightarrow_T s(D_k)$ does not contain inner real vertices. It follows with $s(D_0) \in V(C_i)$ that the parent of D_k is C_i.

Assume to the contrary that D_0 is contained in Exception 41.2. Then D_k is of Type 2b and the set $\{D_0, \dots, D_k\}$ of ancestors of D_0 in H is a bad caterpillar. Let D be the parent of D_k. We claim that $D = C_i$. This implies that D_k is contained in $\mathit{Children}_{12}(C_i)$, which contradicts the assumption that $H \cap \mathit{Children}_{12}(C_i) = \emptyset$.

We prove the remaining claim that $D = C_i$. Assume to the contrary that $D \neq C_i$. As $\{D_0, \dots, D_k\}$ is a bad caterpillar and $s(D_0) \in V(C_i)$, $s(D_0)$ is contained in the intersection of $D \setminus s(D)$ and C_i. We first show that $s(D_0) = t(D)$. If $C_i = C_0$, $D \setminus s(D)$ can intersect with C_0 only at the vertex $t(D)$, which implies that $s(D_0) = t(D)$. Let $C_i \neq C_0$. Then $s(D_0)$ can neither be $s(C_i)$ nor $t(C_i)$, as these vertices are contained in proper ancestors C_j of C_i, contradicting the assumption that $\mathit{Type}_3(C_j) = \emptyset$. Thus, $s(D_0)$ is an inner vertex of C_i. Since D can contain the inner vertex $s(D_0)$ of C_i only as end vertex and because $s(D_0) \in V(D \setminus s(D))$, $s(D_0) = t(D)$ holds. For the same reason, C_i must be the parent of D.

We take this to a contradiction. As $\{D_0, \dots, D_k\}$ is a bad caterpillar, D contains no inner real vertex by definition. Moreover, D cannot be a backedge, as it has the child D_k. Thus, there is a chain C_z of Type 3 that starts at an inner vertex in $t(D) \rightarrow_T s(D)$ due to Lemma 23. This chain C_z is not contained in S_l^R, as otherwise D would contain an inner real vertex. If $s(C_z)$ is contained in C_i, C_z contradicts the choice of C_y, as C_z would be in B and $s(C_z)$ would be a proper ancestor of $s(C_y)$. Otherwise, $s(C_z)$ is contained in a proper ancestor C_j of C_i and contradicts the assumption $\mathit{Type}_3(C_j) = \emptyset$. □

For each of the Exceptions 41.1–41.3 in Lemma 41, the parent of D_k contains a path that

obstructs D_0 and its ancestors from being added, as it contains no inner real vertex. We refer to this path as follows.

Definition 45. Let a chain C_i that is of Type 1 or 2a and has parent C_k be *dependent* on the path $t(C_i) \to_{C_k} s(C_i)$. Let a caterpillar L_i with parent C_k and $s(C_i) \in V(C_k)$ (and every chain contained in it) be *dependent* on the path $s(C_i) \to_{C_k} s(C_k)$. The dependent path of a chain that is of Type 3a or contained in a caterpillar L_i with $s(C_i) \notin V(C_k)$ is defined to be empty.

The idea behind the dependent path P of certain clusters D is that P carries information that is needed to decide whether D is addable. E. g., if the parent of D is contained in S_l^R and P is empty, we can add D, as pointed out by Lemmas 33 and 36. In the case when the parent of D is contained in S_l^R but P is not empty, whether D can be added essentially depends on the existence of inner real vertices in P. We show next that non-empty dependent paths of the minimal chains in segments H of S_l^R separate H from S_l^R.

Lemma 46. *Let H be a segment of S_l^R and let D be the minimal chain of H. If D is neither of Type 3a nor contained in a caterpillar L_i with $s(C_i) \notin V(C_k)$, the dependent path P of D contains all attachment vertices of H.*

Proof. Assume first that D is a chain of Type 1. Then $P = t(D) \to_T s(D)$. If D is a backedge, the claim follows directly. Otherwise, let x be the last inner vertex of D. As T is a DFS-tree and due to Lemma 16, every backedge that enters $T(x)$ starts in P, which gives the claim.

Let C_k be the parent of D. Assume that D is a chain of Type 2a. Since D is a backedge, $s(D)$ and $t(D)$ are the only attachment vertices of H. The claim follows, as the dependent path $P \subset C_k$ of D contains $s(D)$ and $t(D)$.

Due to (R_1) and since C_k is contained in S_l^R, D cannot be of Type 3b. Assume that D is a chain of the remaining Type 2b. According to (R_1), D is contained in a caterpillar L_i, every chain of which is contained in H. By assumption, $s(C_i) \in V(C_k)$. The chain D is not a backedge, as it has a child; let x be the last inner vertex of D. By the construction of caterpillars, C_i is the minimal chain of Type 3 that enters $T(x)$. It follows with Lemma 16 that every backedge that enters $T(x)$ starts either at $s(C_k)$ or at a descendant of $s(C_i)$ in C_k. Thus, all attachment vertices are contained in $P = s(C_i) \to_{C_k} s(C_k)$. □

The following theorem leads to an existence proof of the restricted construction sequence if G is 3-connected.

Theorem 47. *Assume that the input graph G is 3-connected. Let C_i be a chain in S_l^R such that $\text{Children}_{12}(C_j) = \text{Type}_3(C_j) = \emptyset$ holds for every proper ancestor C_j of C_i. Then there is an order σ in which the clusters of all ancestors of the chains in $\text{Children}_{12}(C_i) \cup \text{Type}_3(C_i)$ that are not contained in S_l^R are successively addable.*

Proof. By applying Lemma 44 in advance, we can assume that the segment of every chain in $\mathit{Type}_3(C_i)$ contains a chain in $\mathit{Children}_{12}(C_i)$. Let H be such a segment of a chain in $\mathit{Type}_3(C_i)$ and let D_k be the minimal chain in H. According to Lemma 40, D_k is the only chain in H that is in $\mathit{Children}_{12}(C_i)$.

If $\mathit{Children}_{12}(C_i) = \emptyset$, $\mathit{Type}_3(C_i) = \emptyset$ and the claim follows. We will show that $\mathit{Children}_{12}(C_i) \neq \emptyset$ causes $\mathit{Children}_{12}(C_i)$ to contain a chain D whose cluster is addable. Assume for a moment that this already has been shown and let H' be the segment of D. Adding the cluster of D to the current

subgraph deletes D from the set $\mathit{Children}_{12}(C_i)$. Therefore, H' does not contain a minimal chain in $\mathit{Children}_{12}(C_i)$ anymore and we can add the clusters of all ancestors of the chains in $\mathit{Type}_3(C_i) \cap H'$ that are contained in H' by applying Lemma 44. Iterating this argument gives the claim.

Assume to the contrary that $\mathit{Children}_{12}(C_i) \neq \emptyset$ and that the cluster of every chain in $\mathit{Children}_{12}(C_i)$ is not addable. We first subsume properties of every chain $D \in \mathit{Children}_{12}(C_i)$; let H be the segment of D. By definition, D is of Type 1 or 2; D is also the minimal chain in its segment H with Lemma 40. Let D be of Type 1. Then the dependent path P of D does not contain an inner real vertex, as otherwise D is addable with Lemma 33. According to Lemma 23, D must be either a backedge or H contains a chain of Type 3 that starts in C_i, implying that H contains a chain in $\mathit{Type}_3(C_i)$. In the latter case, we can apply Lemma 42 and it follows that D must be contained in Exception 41.1 or 41.3 (as the chain D_k).

Let D be of Type 2. Then $C_i \neq C_0$. If D is of Type 2a, neither $t(D)$ nor an inner vertex in the dependent path P of D can be real, since otherwise D is addable due to Lemma 33. If D is of Type 2b, D is the minimal chain of a caterpillar with parent C_i. As we assumed that the cluster of D is not addable, this caterpillar must be bad with Lemma 36 and there is no inner real vertex in the dependent path P of D. Because $H \cap \mathit{Type}_3(C_i) \neq \emptyset$, it follows with Lemma 42 that D is contained in Exception 41.2 (as the chain D_k). We list the possible cases for each D.

1. D is of Type 1 without an inner real vertex in P and either a backedge or the chain D_k in Exception 41.1 or 41.3
2. $C_i \neq C_0$ and D is of Type 2a without a real vertex in $P \setminus s(D)$
3. $C_i \neq C_0$, D is of Type 2b without an inner real vertex in P and D is the chain D_k in Exception 41.2

In each of these cases, P is contained in C_i. If D is a backedge (possible in Cases 1. and 2.), P is of length at least two, as G is simple. If D is no backedge (possible in Cases 1. and 3.), P is also of length at least two due to Lemma 39. In every case, P does not contain an inner real vertex. Hence, the dependent path P for some chosen chain D is contained in a link L of S_l^R. Thus, $P \subseteq L \subseteq C_i$ and L is of length at least two. According to Lemma 9 and the 3-connectivity of G, a parallel link of L (maybe L itself) contains an inner vertex v on which a BG-path starts. Note that this BG-path does neither have to be a chain nor preserve (R_1) or (R_2). Also, v is not necessarily contained in P.

We show next that L itself contains v as an inner vertex due to our imposed restrictions on the construction sequence. This will imply a contradiction later. Assume first that all edges of L are DFS-tree edges. If $S_l^R = S_3^R$, $L = C_0$ must hold. Because G is simple, at least one chain of C_1 and C_2 is not a backedge, say C_1. Let x be the last inner vertex of C_1. The claim follows by applying Lemma 23 on C_1, as the chain of Type 3 that enters $T(x)$ can be extended to a BG-path ending at an inner vertex of C_1. If $S_l^R \neq S_3^R$, the claim follows from L having no different parallel link due to Restriction (R_2) in S_l^R.

Now assume that L contains a backedge. Since L is contained in C_i, $s(C_i)$ must be an end vertex of L. Let w be the other end vertex of L. As C_i has a child, C_i is no backedge. Applying Lemma 23 on C_i gives a chain of Type 3 that starts at a proper ancestor C_j of C_i and enters $T(x)$ for x being the last inner vertex of C_i. As $\mathit{Type}_3(C_j) = \emptyset$ holds by assumption and S_l^R is upwards-closed, C_i contains an inner real vertex. Therefore, w is different from $t(C_i)$. If there is a parallel link $L' \neq L$ of L in S_l^R, let C_k be the chain that contains L'. Then $s(C_i) = s(C_k)$, $C_k = L'$

and C_k is a child of C_i, implying that C_k is of Type 2. The chain C_k is not of Type 2b, as otherwise C_k would contain an inner real vertex due to the caterpillar $L_k \subset S_l^R$. Therefore, C_k is of Type 2a and cannot contain an inner vertex, as it is a backedge. We conclude that v is an inner vertex of L on which a BG-path B starts.

Let H be the segment of S_l^R that contains B. Assume first that H contains a chain $D' \in \mathit{Children}_{12}(C_i)$ and let P' be the dependent path of D'. Then all attachment vertices of H must be contained in P' by Lemma 46, this holds in particular for v. As v is non-real and H is not contained in S_l^R, B must contradict Property 4.2. It follows that H does not contain any child of C_i that is of Type 1 or 2. Because $\mathit{Type}_3(C_j) = \emptyset$ for all proper ancestors C_j of C_i, H cannot contain a child of C_i that is of Type 3. We conclude that H does not contain any child of C_i.

Assume that H contains a chain C_a of Type 1 with $s(C_a) = v$. Let C_b be the parent of C_a, as $C_a \neq C_0$. Note that C_b is not necessarily in H. Then $C_b \neq C_i$, as otherwise C_a would be a child of C_i in H. By definition of Type 1, C_b contains the tree edges $t(C_a) \to_T v$ and $t(C_b) = v$ follows, as v is contained in C_i. This implies that C_b is a child of C_i, giving a contradiction. Additionally, H cannot contain a chain C_a of Type 3 with $s(C_a) = v$, as otherwise C_a would be contained in $\mathit{Type}_3(C_i)$, which implies that H contains a chain in $\mathit{Children}_{12}(C_i)$. Let J be the chain that contains the first edge of B. By the previous arguments, J must be of Type 2 and $s(J) = v$.

We take this to a contradiction. Let C_a be the maximal (with respect to $<$) ancestor of J that is not of Type 2. Then $s(C_a) = v$ holds by definition of Type 2 and C_a must be contained in H, as $s(C_a)$ is non-real. This contradicts that H contains neither a chain of Type 1 nor a chain of Type 3 that starts at v. It follows that B cannot exist, implying the claim. □

Assume for a moment that G is 3-connected. In order to be able to apply Theorem 47, we show that every S_l^R contains a chain C_i that satisfies the preconditions of Theorem 47. This is clearly the case in S_3^R, as then C_0 is such a chain. Applying Theorem 47 on C_i and adding the clusters in σ generates a subgraph S_t^R. The subgraph S_t^R must contain all children of C_i and therefore causes the precondition to hold for C_{i+1}. This ensures that iteratively applying Theorem 47 on $C_0, C_1, \ldots, C_{m-n+1}$ constructs G, which proves the existence of the restricted construction sequence.

Corollary 48. Let G be a 3-connected graph with a chain decomposition $C = \{C_0, \ldots, C_{m-n+1}\}$. Then there is a construction sequence of G restricted by (R_1) and (R_2) that starts with $S_3^R = \{C_0 \cup C_1 \cup C_2\}$.

5.7 The Algorithm

We already described the parts of the certifying algorithm that

- compute the DFS-tree T,
- compute whether G has connectivity 0, 1 or at least 2,
- compute the chain decomposition C and the chain classification,
- check whether Properties A and B are satisfied and
- compute S_3^R.

All steps can be computed in time $O(n+m)$ (see Sections 5.1–5.3).

Although G is not known to be 3-connected, the proof of Theorem 47 still provides an algorithmic method to search iteratively for BG-paths that build the restricted construction sequence, starting with S_3^R: We iterate over all chains C_i, $0 < i < m-n+1$ (we say that C_i is *processed*). For C_i, let B be the set of all ancestors of the chains in $\mathit{Children}_{12}(C_i) \cup \mathit{Type}_3(C_i)$ that are not contained in the current subgraph. We try to find an order on the clusters of B in which they are all addable. This order does not necessarily exist, as G might not be 3-connected. However, we will give an algorithm that either computes such an order or finds a separation pair. In the case that this order can be found for every C_i, we obtain a construction sequence of G, which certifies G to be 3-connected.

During the chain classification, we store on each chain a list of its children of Type 1 and 2 and on each vertex v a list of the chains of Type 3 that start at v. This allows us to compute the sets $\mathit{Children}_{12}(C_i)$ and $\mathit{Type}_3(C_i)$ efficiently for each C_i in time $O(|C_i| + |\mathit{Children}_{12}(C_i)| + |\mathit{Type}_3(C_i)|)$ by traversing $(t(C_i) \to_{C_i} s(C_i)) \setminus s(C_i)$.

We describe the processing phase of C_i. First, the sets $\mathit{Children}_{12}(C_i)$ and $\mathit{Type}_3(C_i)$ are computed. For convenience, we perform the following test in advance (this can however be handled differently in an implementation). For each $D \in \mathit{Children}_{12}(C_i)$ that is of Type 2a and for which $t(D)$ is real, we add D due to Lemma 33.

Let S_l^R be the current subgraph. We partition the chains in $\mathit{Type}_3(C_i)$ into the segments of S_l^R by storing a pointer on each $C_j \in \mathit{Type}_3(C_i)$ to the minimal chain D of the segment that contains C_j. The chain D is computed by traversing the path in T from $t(C_j)$ to the root of T until we encounter a vertex v with a parent that is already contained in S_l^R. Then v must be an inner vertex of D (each inner vertex has a pointer to its chain) and we mark each vertex of the traversed path with "D". Further traversals in the same segment stop at the first vertex that is marked; hence, they will find the marker "D". Since the clusters of all traversed chains will eventually be added in the same processing phase, the total running time of these traversals for all processed chains adds up to a total of $O(m)$.

Let X be the subset of the chains in $\mathit{Type}_3(C_i)$ that are contained in segments in which the minimal chain is not a chain of $\mathit{Children}_{12}(C_i)$. With Lemma 40, each such segment does not contain any chain in $\mathit{Children}_{12}(C_i)$. To compute X, it suffices to check for each $C_j \in \mathit{Type}_3(C_i)$ in constant time whether its pointer points to a chain in $\mathit{Children}_{12}(C_i)$. In the affirmative case, C_j is not contained in X; otherwise, C_j is contained in X.

According to Lemma 44, the clusters of all ancestors of the chains in X that are not in S_l^R are successively addable. Moreover, these clusters are successively addable in the order in which they were traversed when partitioning the chains of $\mathit{Type}_3(C_i)$ into segments (i. e., in ascending order of $<$). We add them in this order. However, as these clusters form a subtree U' of U for each segment H with $H \cap X \neq \emptyset$, any other pre-order on $V(U')$ in conformance with Lemma 44 can be computed in time linearly dependent on $|V(U')|$. Whenever a cluster containing a chain in $\mathit{Type}_3(C_i)$ is added, we delete it from $\mathit{Type}_3(C_i)$.

Let S_t^R be the generated graph. The segment H of every remaining chain in $\mathit{Type}_3(C_i)$ must contain a chain D in $\mathit{Children}_{12}(C_i)$. The chain D is the minimal chain in H due to Lemma 40. According to Theorem 47, the clusters of all ancestors of the chains in $\mathit{Children}_{12}(C_i) \cup \mathit{Type}_3(C_i)$ that are not in S_t^R are successively addable if G is 3-connected. However, Theorem 47 does not specify in which order these clusters are successively addable, so we need to compute a valid order on them (if this order exists).

Let H be the segment of a chain D in $\mathit{Children}_{12}(C_i)$. The cluster of D cannot be a caterpillar L_j with $s(C_j) \notin V(C_i)$, as we maintain the invariant that $\mathit{Children}_{12}(C_k) = \mathit{Type}_3(C_k) = \emptyset$ for every proper ancestor C_k of C_i. Thus, D is dependent on a non-empty path $P \subseteq C_i$ by Definition 45. This path contains all attachment vertices of H with Lemma 46. We can compute the attachment vertices of H efficiently, as the previously described partition of $\mathit{Type}_3(C_i)$ into segments provides the set $H \cap \mathit{Type}_3(C_i)$. The attachment vertices of H are the union of the start vertices of the chains in $H \cap \mathit{Type}_3(C_i)$ and the vertices $s(D)$ and $t(D)$. This gives also P, as P is the path of maximal length in C_i that has attachment vertices of H as end vertices.

For computing a possible order in which the remaining clusters are addable, we need the following lemma. It shows that the clusters of all ancestors of the chains in $H \cap (\mathit{Children}_{12}(C_i) \cup \mathit{Type}_3(C_i))$ that are not contained in S_t^R are addable if and only if P contains an inner real vertex.

Lemma 49. *Let D be a remaining chain in $\mathrm{Children}_{12}(C_i)$ in S_t^R, let H be the segment of S_t^R that contains D and let P be the dependent path of D. If P contains an inner real vertex, the clusters of all ancestors of the chains in $H \cap (\mathrm{Children}_{12}(C_i) \cup \mathrm{Type}_3(C_i))$ that are not in S_t^R are successively addable. Moreover, these clusters are addable in any pre-order that adds clusters that start at $t(D)$ last, e. g., in ascending order of $<$. Conversely, if P does not contain an inner real vertex, no cluster in H is addable.*

Proof. Let P contain an inner real vertex. If $H \cap \mathit{Type}_3(C_i) \neq \emptyset$, the claim follows directly from Lemma 42, as the dependent path of the minimal chain (D_k) of each of the Exceptions 41.1–41.3 contains no inner real vertex. If $H \cap \mathit{Type}_3(C_i) = \emptyset$, we only need to show that the cluster of D is addable. Moreover, D must be of Type 1 or 2a, as otherwise $H \cap \mathit{Type}_3(C_i) \neq \emptyset$. According to Lemma 33, D is addable.

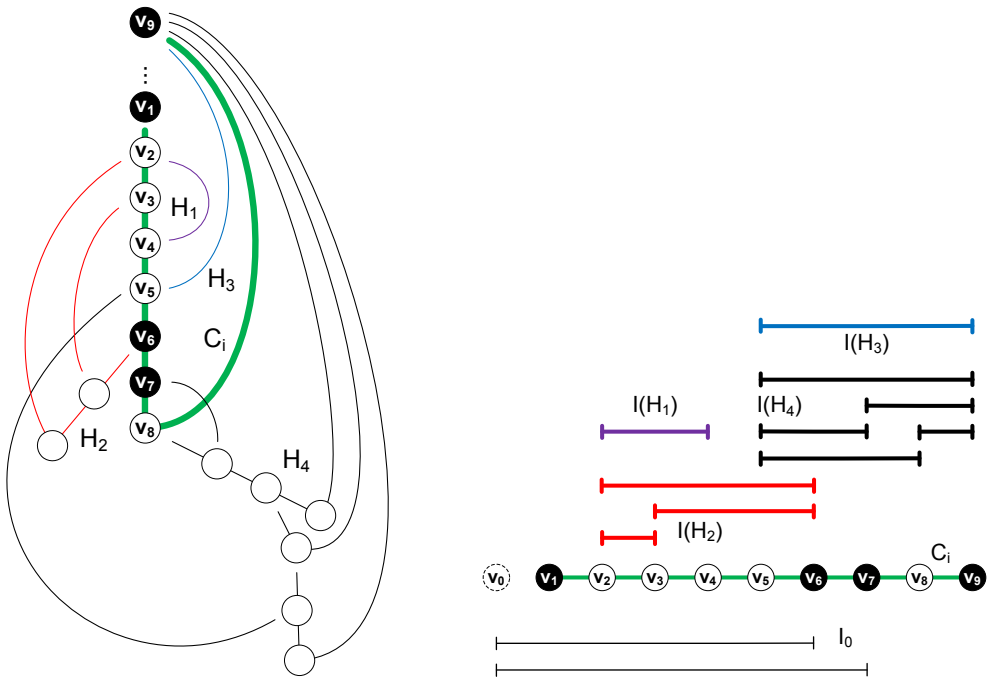
Otherwise, P does not contain an inner real vertex. Assume to the contrary that we can add a cluster in H. As Restriction (R_1) requires upwards-closedness after adding each cluster, the cluster of D must be added first. According to Lemma 46, all attachment vertices of H are contained in P. The end vertices of P must be real, as otherwise no path in H can satisfy Property 4.2. Since $D \in \mathit{Children}_{12}(C_i)$, D cannot be of Type 3. If D is of Type 1, $s(D)$ and $t(D)$ must be the end vertices of P, P consists only of tree edges and adding D would contradict Restriction (R_2).

Therefore, D is of Type 2. Then P must contain the backedge in C_i and it follows that $s(D) = s(C_i)$. If D is of Type 2a, the end vertices of D are the end vertices of P. Since both end vertices are real, D would have been added before with Lemma 33.

We conclude that D is of Type 2b. Then the cluster of D is a bad caterpillar L_j, as $s(C_j) \notin V(C_i)$ would contradict $\mathit{Type}_3(C_k) = \emptyset$ for a proper ancestor C_k of C_i and because P does not contain an inner real vertex. Let Q_1 and Q_2 be the first two BG-paths in L_j that are added as part of the addition of L_j. As Q_1 must connect the two end vertices of P and since L_j contains exactly one edge e that is incident to $s(C_j)$, Q_1 contains e. Thus, Q_2 cannot contain the end vertex $s(C_j)$ of P and it follows that Q_2 has at most one real end vertex. Since P and Q_1 are parallel links of S_{t+1}, Q_2 violates one of the Properties 4.1–4.3. This contradicts Restriction (R_1), as L_j, the cluster of D, can not be decomposed into BG-paths. □

5.7.1 Reduction to Overlapping Intervals

Let Y be the set of segments that contain a remaining chain in $\mathit{Children}_{12}(C_i)$. Let the *dependent path* of a segment $H \in Y$ be the dependent path of its minimal chain, i. e., the maximal path in



(a) The chain C_i and the remaining chains in $\mathit{Type}_3(C_i) \cup \mathit{Children}_{12}(C_i)$.

(b) For each of the two inner real vertices v_6 and v_7 in C_i, there is one interval in I_0.

Figure 11: Mapping segments to intervals. Different colors depict different segments.

C_i connecting two attachment vertices of H. Let an order σ on a subset of Y be *proper* if the dependent path of each segment in σ contains an inner real vertex or an inner vertex that is an attachment vertex of a previous segment in σ. Note that the addition of the clusters in previous segments H would cause the attachment vertices of H to be real.

Let B be all ancestors of chains in $\mathit{Children}_{12}(C_i) \cup \mathit{Type}_3(C_i)$ that are not in S_l^R. According to Lemma 49, finding an order in which the clusters of B are successively addable reduces to finding a proper order σ on Y. Having σ allows us to add the clusters of $B \cap H$ subsequently for each segment H in σ. We show how a proper order σ on Y can be efficiently computed by a reduction to overlaps of intervals.

A very clear and simple characterization of 3-connectivity in terms of a binary relation (called *directly-linked*) on the segments of cycles is given by Vo [67, 68] and based on the work of Williamson [70]. The binary relation represents a graph whose connectivity determines the 3-connectivity of the input graph. To compute the proper order σ on Y (if it exists), we will use a similar concept: We reduce the computation of σ to the computation of a spanning tree of the graph G' of a certain binary relation. In particular, σ exists if G' is connected. The binary relation will correspond to overlaps on intervals. The structure imposed by previous computation steps allows us to compute these intervals efficiently.

We map each segment $H \in Y$ to a set $I(H)$ of intervals on the elements of $V(C_i)$: Let $a_1, \ldots, a_k$ be the attachment vertices of H and let $I(H) = \bigcup_{1<j\leq k}\{[a_1, a_j]\} \cup \bigcup_{1<j<k}\{[a_j, a_k]\}$ (see Figure 11). Additionally, we augment $V(C_i)$ by an artificial first vertex v_0 and map the real vertices $b_1, \ldots, b_k$ of C_i to the set of intervals $I_0 = \bigcup_{1<j<k}\{[v_0, b_j]\}$. This construction can be efficiently computed

and creates at most $|Children_{12}(C_i)| + 2|Type_3(C_i)| + |V_{real}(C_i)| - 2$ intervals for C_i, adding up to a total of $O(m)$ intervals for all chains.

Let two intervals $[a, b]$ and $[c, d]$ *overlap* if $a < c < b < d$ or $c < a < d < b$. We want to compute a proper order on Y by finding a sequence of overlapping intervals starting with an interval in I_0. Let the *overlap graph* of Y be the graph with vertex set $I_0 \cup \bigcup_{H \in Y} I(H)$ and an edge between two vertices if and only if the corresponding intervals overlap. Let the *merged overlap graph* of Y be the graph that results from the overlap graph by merging the vertices corresponding to I_0 and to $I(H)$ for every segment $H \in Y$, respectively, to one vertex.

Lemma 50. *There is a proper order on the segments in Y if and only if the merged overlap graph G' of Y is connected.*

Proof. Let G' be connected and G'' be a spanning connected subgraph of G'. Then $V(G') = V(G'') = Y \cup \{I_0\}$. Let $H_0, H_1, \ldots, H_{|Y|}$ be the order in which the vertices of G'' are visited first by an arbitrary DFS in G'' that starts on $I_0 = H_0$. We show that $\sigma = H_1, \ldots, H_{|Y|}$ is a proper order on Y. Let H_i, $1 \leq i \leq |Y|$, be a segment in σ. Then H_i is adjacent to a vertex H_j, $j < i$, in G'' by construction and an interval in $I(H_i)$ overlaps with an interval in $I(H_j)$. If $j = 0$, the dependent path of H_i contains an inner real vertex. If $j > 0$, the dependent path of H_i contains an inner vertex that is an attachment vertex of the previous segment H_j of σ.

Let $\sigma = H_1, \ldots, H_{|Y|}$ be a proper order on Y. We show that every H_i, $1 \leq i \leq |Y|$, is adjacent to I_0 or a vertex H_j with $j < i$ in G'. This implies that G' is connected. If the dependent path P_i of H_i contains an inner real vertex v, v must be the end point of an interval $[v_0, v]$ in I_0. Since P_i does not contain v_0, the interval $[s(P_i), t(P_i)]$ in $I(H_i)$ and $[v_0, v]$ overlap. Otherwise, P_i does not contain an inner real vertex. In that case, P_i contains an inner vertex that is an attachment vertex of a segment H_j with $1 \leq j < i$, as σ is proper.

Let v be an inner vertex of P_i that is an attachment vertex of the segment H_j with minimal j and let P_j be the dependent path of H_j. If $P_j \subseteq P_i$, no inner vertex of P_j can be real and it follows that P_j contains an inner vertex that is an attachment vertex of a segment H_k with $1 \leq k < j$, contradicting the choice of v. Thus, P_j is not contained in P_i. We conclude that $I(H_j)$ contains an interval $[v, u]$ or $[u, v]$ with u being an end vertex of P_j that is not in P_i. This implies that $[v, u]$ or $[u, v]$ overlaps with the interval $P_i \in I(H_i)$. □

If G is 3-connected, a proper order σ on Y exists according to Theorem 47 and Lemma 49. The merged overlap graph G' of Y is then connected with Lemma 50. The following algorithm detects whether G' is connected and computes σ in the affirmative case and the connected components of G' otherwise.

Lemma 51. *Let t be the total number of intervals that have been created for I_0 and all segments in Y and let G' be the merged overlap graph of Y. There is an algorithm with running time $O(t + |V(C_i)|)$ that computes a proper order σ on Y, if exists, and that computes the connected components of G', if no proper order on Y exists.*

Proof. We may construct G' explicitly and, if G' is connected, extract σ from G' as described in the proof of Lemma 50. Unfortunately, then G' cannot be computed explicitly in time linearly dependent on t, as it can contain up to $\binom{t}{2} \in \Omega(t^2)$ edges. For a worst case example, consider t slightly shifted copies of the same interval.

We cope with this problem by computing a *sparse* merged overlap graph, i. e., a spanning subgraph G_s of G' with at most $2t$ edges but with exactly the same connected components as G'.

A simple sweep-line algorithm due to Olariu and Zomaya [45] computes in time $O(t)$ a spanning subgraph G'_s of the overlap graph of Y (but not of the *merged* overlap graph) such that G'_s has at most $2t$ edges and exactly the same connected components as the overlap graph of Y. The algorithm assumes the end points of intervals to be sorted. As we deal only with intervals that correspond to vertices on a chain C_i and an extra vertex v_0, we can apply bucket sort in advance to sort the end points. We remark that this will not be necessary in the application of this lemma, as predecessors and successors in this order can be maintained during a traversal of C_i. Note also that the work in [45] describes mainly a non-sequential variant of the algorithm; for a simple sequential variant, Lemmas 4.1 and 4.2 in [45] suffice.

Thus, G'_s can be computed in time $O(t + |V(C_i|)$. To obtain G_s, we just have to merge the sets I_0 and $I(H)$ for each $H \in Y$ in G'_s to one vertex, respectively. This takes a total running time of $O(t + |V(C_i|)$. In G_s, we apply a DFS on the vertex I_0 to decide whether G_s is connected. If G_s is connected, G_s is a spanning connected subgraph of G'. Then, as described in the first lines of the proof of Lemma 50, the order in which the vertices are visited first in the DFS gives a proper order on Y. If G_s is not connected, the DFS computes the connected components of G_s, which are exactly the connected components of G'. □

For every chain $C_i \in C$, at most $|\mathit{Children}_{12}(C_i)| + 2|\mathit{Type}_3(C_i)| + |V_{\mathit{real}}(C_i)| - 2$ intervals are created. Thus, applying the algorithm of Lemma 51 for all chains in C adds up to a total time of $O(m)$.

We therefore gave a linear-time algorithm that either computes a construction sequence of G or returns a witness for the fact F that there is no proper order σ on Y, namely that more than one connected component of the merged overlap graph G' exists. According to Theorem 47 and Lemma 50, F proves that G is not 3-connected. However, to obtain an easy-to-verify certificate in that case, we will show how a separation pair can be extracted in the next section. If the input graph is assumed to be 3-connected, we get the following theorem.

Theorem 52. *The sequences* (1)–(8) *for a simple* 3*-connected graph* G *can be computed in time* $O(m)$.

We remark that the reduction to intervals does not have to be explicit in an implementation; instead, we can work directly on the graph.

5.7.2 Extracting a Separation Pair

We show how a separation pair can be extracted if not all clusters of $\mathit{Children}_{12}(C_i) \cup \mathit{Type}_3(C_i)$ have been added after processing C_i. Then $\mathit{Children}_{12}(C_i)$ must still contain a chain D. Let H be the segment that contains D and let S_l^R be the current subgraph. Let W be the union of H and every segment that is mapped to the same connected component A of the merged overlap graph as H. The set W can be represented as the connected component A itself, which was already computed in the processing phase of C_i with the algorithm of Lemma 51.

The union of dependent paths of the segments in W is a path $P = x \to y$ that is contained in C_i, as there is a sequence of overlapping intervals for A. The path P cannot contain inner vertices that are real due to Lemma 49. It follows that every edge in $E(G) \setminus E(P)$ that is adjacent to an inner vertex in P must be contained in a segment H' of S_l^R (H' does not have to be contained in W). As H' is contained in W or the intervals of H' do not overlap with any interval of a chain in W, all the attachment vertices of H' must be in P. Additionally, P contains an inner vertex v,

since the two attachment vertices of segments that are backedges cannot connect two consecutive vertices in T, as G is simple, and because every other segment has at least three attachment vertices due to Lemma 39.

Therefore, deleting x and y separates v from $G \setminus V(P)$ and x and y form a separation pair, certifying that G is not 3-connected. The vertices x and y can be computed in $O(n+m)$ by considering the attachment vertices of the segments in W. This gives the following result.

Theorem 53. *There is a certifying algorithm that tests the* 3*-connectivity of a simple graph* G *in time* $O(n+m)$*, returns each of the sequences* (1)–(8) *if* G *is* 3*-connected and returns a cut vertex or a separation pair otherwise.*

Algorithm 2 gives an overview of the whole approach. An example of its application can be found in Figure 12. We get the following corollary from the discussion of certificates for 3-edge-connectivity (see Section 4.1) and edge cuts (see Section 4).

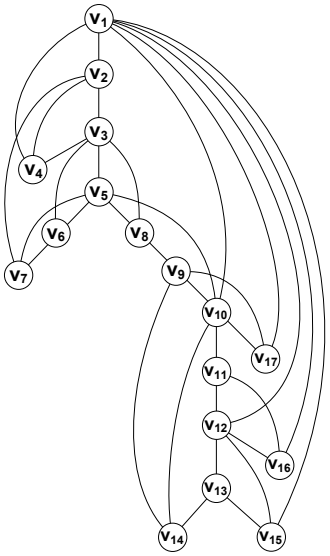
Algorithm 2 Test3Connectivity(Graph G)

```
Compute a DFS-tree T of G, a chain decomposition C and classify the chains
Check Properties A and B, Compute S_3^R := C_0 ∪ C_1 ∪ C_2          ▷ Sections 5.1–5.3
for i := 0 to m − n + 1 do                                          ▷ process each C_i
    Compute the lists Children_12(C_i) and Type_3(C_i)              ▷ page 33
    for each D ∈ Children_12(C_i) that is of Type 2a such that t(D) is real do
        Add D; update Children_12(C_i)          ▷ omissible, see Section 5.7.3
    Partition Type_3(C_i) into segments                             ▷ page 33
    (every such segment is represented by the minimal chain it contains)
    for each segment H with H ∩ Type_3(C_i) ≠ ∅ and H ∩ Children_12(C_i) = ∅ do
        Add the clusters of all ancestors of H ∩ Type_3(C_i) that are in H in the
        order of <; update Type_3(C_i)                              ▷ Lemma 44
    Let Y be the set of segments that contain a chain in Children_12(C_i)
    for each H ∈ Y do
        Compute the attachment vertices and the dependent path of H ▷ page 34
        Map H to a set of intervals on C_i                          ▷ Section 5.7.1
    Using these intervals, compute either a proper order σ on Y or more than
    one connected component of the merged overlap graph G' of Y     ▷ Lemma 51
    if a proper order σ was computed then
        for each segment H ∈ Y in the order of σ do   ▷ Add clusters; save construction seq.
            Add the clusters of all ancestors of H ∩ (Type_3(C_i) ∪ Children_12(C_i))
            that are in H in the order of <; update Type_3(C_i)     ▷ Lemma 49
    else                                              ▷ G' is not 3-connected
        Compute a separation pair                                   ▷ Section 5.7.2
```

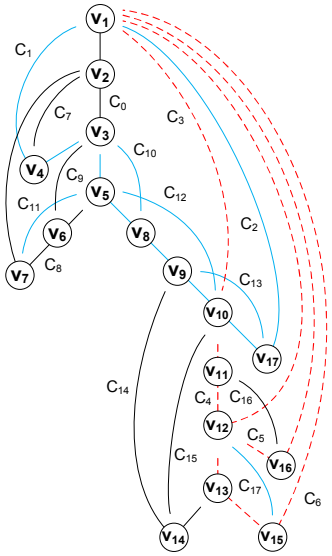
Corollary 54. There is a certifying algorithm that tests the 3-edge-connectivity of a simple graph G in time $O(n+m)$.

5.7.3 Simplifications

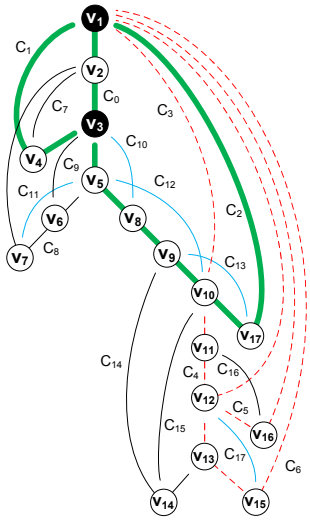
We list some simplifications for Algorithm 2.



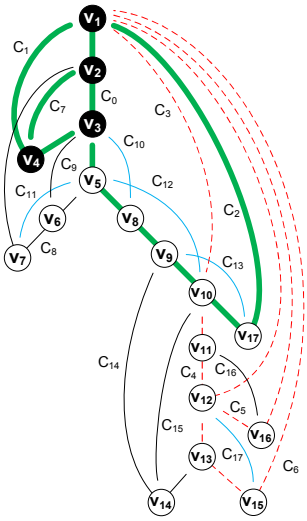
(a) A 3-connected input graph G with $n = 17$ and $m = 33$. Straight lines depict the edges of the DFS-tree T (the DFS visits left children first).



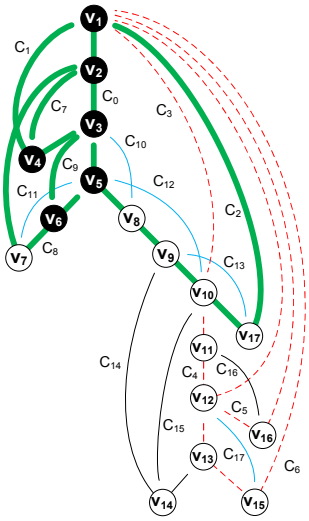
(b) The decomposition of G into $m - n + 2$ chains. Light solid chains are of *Type 1*, red dashed ones of *Type 2* and black solid ones of *Type 3*. The only chain of Type 2a is C_3. The only chains of Type 3b are C_{14} and C_{16}, giving the caterpillars $L_{14} = \{C_{14}, C_6, C_4\}$ and $L_{16} = \{C_{16}, C_5\}$, respectively.



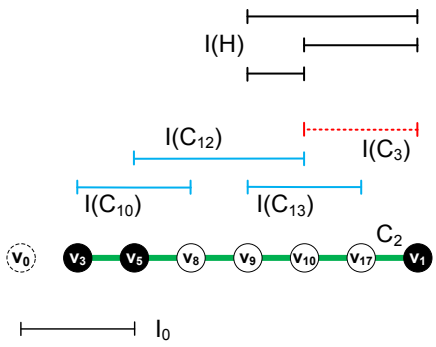
(c) The K_2^3-subdivision $S_3^R = \{C_0, C_1, C_2\}$ in G (thick green edges). We start with processing C_0. Since $\mathit{Children}_{12}(C_0) = \emptyset$, we can add all chains in $\mathit{Type}_3(C_0) = \{C_7, C_8, C_9\}$ with Lemma 44 in ascending order of $<$. Thus, we add C_7 first.



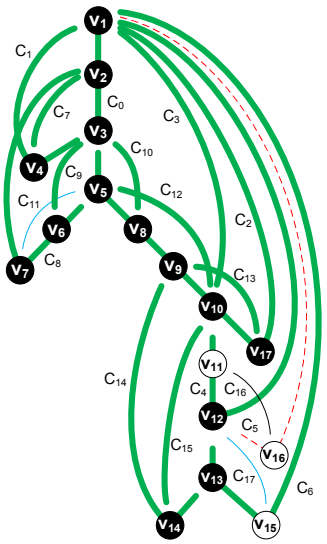
(d) The K_4-subdivision S_4^R that is generated by the addition of C_7. Its real vertices are depicted in black. Note that choosing C_8 instead of C_7 would have led to a K_4-subdivision as well. The remaining chains C_8 and C_9 are added in that order.



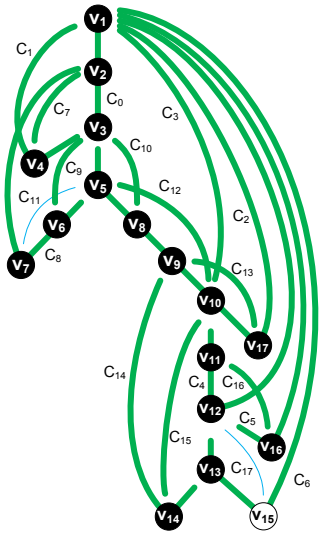
(e) The subgraph S_6^R. As C_1 is finished, we process C_2 next. Due to Theorem 47, C_3, C_{10}, C_{12}, C_{13}, C_{15} and L_{14} are addable if G is 3-connected. To obtain the right order on these clusters, we group them by segments and compute the following mapping.



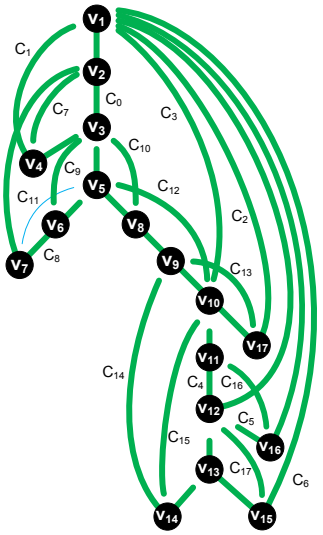
(f) Mapping segments to intervals (I_0 is induced by v_5). Let H be the segment of C_{14}. Any sequence of overlapping intervals, e. g., I_0, $I(C_{10})$, $I(C_{12})$, $I(C_{13})$, $I(C_3)$, $I(H)$, gives an order on segments. For each such segment H', we add the clusters of all ancestors of $H' \cap (\mathit{Children}_{12}(C_2) \cup \mathit{Type}_3(C_2))$ in H'.



(g) The subgraph S^R_{14} after adding the clusters C_{10}, C_{12}, C_{13}, C_3, L_{16} and C_{15} due to Lemma 49. The next non-trivial chain to process is C_4 with $\mathit{Children}_{12}(C_4) = \{C_5\}$ and $\mathit{Type}_3(C_4) = \{C_{16}\}$. Both chains are contained in the good caterpillar L_{16}, which is a segment of S^R_{14} itself. We create the interval $[v_0, v_{12}]$ for C_4 due to its inner real vertex v_{12} and map L_{16} to the intervals $[v_{11}, v_1]$, $[v_{11}, v_{12}]$ and $[v_{12}, v_1]$ on C_4. As $[v_0, v_{12}]$ overlaps with $[v_{11}, v_1]$, we can add the caterpillar L_{16}, which is decomposed into the two BG-paths $v_{11} \to_{G \setminus E(S^R_{14})} v_1$ and $v_{17} \to_T v_{12}$ with Lemma 36.



(h) The subgraph S^R_{16}. As there is nothing to do for C_5, we next process C_6 with $\mathit{Children}_{12}(C_6) = \{C_{17}\}$ and $\mathit{Type}_3(C_6) = \emptyset$. Due to the inner real vertex v_{13} of C_6 (causing an interval of I_0 to overlap with $I(C_{17})$), C_{17} is addable.



(i) The subgraph S^R_{17}. The next non-trivial chain to process is C_8 with $\mathit{Children}_{12}(C_8) = \{C_{11}\}$ and $\mathit{Type}_3(C_8) = \emptyset$. Due to the inner real vertex v_6 of C_8 (causing an interval of I_0 to overlap with $I(C_{11})$), C_{11} is addable as the last BG-path of the construction sequence. This generates the subgraph S^R_{18}, which is identical to G.

Figure 12: An example of the algorithm.

- Treatment of chains of Type 2a:
  Lines 5 and 6 in Algorithm 2 can be omitted. Let C_i be the currently processed chain and suppose there is a chain $D \in \mathit{Children}_{12}(C_i)$ of Type 2a with $t(D)$ being real. We show that D will be added if G is 3-connected (if G is not 3-connected, the absence of D does not harm the algorithm to find a separation pair, as D is a backedge).

  Let G be 3-connected and let P be the dependent path of D. We can assume that P has no inner real vertex after processing C_i, as otherwise D would have been added as well by construction of the merged overlap graph. Clearly, C_i must contain an inner vertex, since D is a child of C_i; let v be the first inner vertex of C_i. Since G is simple, $t(D) \neq v$ holds, which implies that v is an inner vertex of P. Thus, v is non-real and it follows that G contains no chain of Type 3 that ends at v. Every chain that starts at v implies that v has a child in T. We conclude with $\mathit{deg}(v) \geq 3$ in G that at least one child of C_i ends at v and is of Type 1 or 2. Since this chain is contained in $\mathit{Children}_{12}(C_i)$ and G is 3-connected, v is real after processing C_i, contradicting the assumption.

- Integrating the preprocessing phase:
  The processing phases of chains can be integrated into the chain decomposition. A chain C_i can already be processed when it is contained in the current subgraph and when all chains of $\mathit{Children}_{12}(C_i) \cup \mathit{Type}_3(C_i)$ have been found. Therefore, the chain decomposition does not have to be finished for computing a proper ordering and for adding clusters. This gives a two-step approach of the whole algorithm: The first step just performs a DFS and the second step computes the construction sequence by processing iteratively $C_0, C_1, \ldots, C_{m-n+1}$.

6 Conclusions

We proposed several variants of construction sequences for the class of 3-connected graphs and developed efficient transformations between them. This led to a conceptually new approach to compute these construction sequences bottom-up, which runs in optimal linear time. Using construction sequences as certificates for 3-connectivity allowed to create certifying algorithms for testing graphs on being 3-connected and 3-edge-connected in time $O(n + m)$.

An $SPQR$-tree of a 2-connected graph G is a tree that represents the 3-connected components of G. Hopcroft and Tarjan [30] and Gutwenger and Mutzel [27] show that the $SPQR$-tree of a 2-connected graph can be computed in linear time. Clearly, we can certify the 3-connected components (the so-called R-nodes) of this tree to be 3-connected with the algorithm of Section 5. In fact, this allows to certify the $SPQR$-tree decomposition itself, which gives a linear-time certifying algorithm for computing $SPQR$-trees. However, it would be interesting and of practical interest whether the algorithm itself can be extended to give an $SPQR$-tree in linear time.

In contrast to the non-certifying algorithms in [30, 67, 68], the given algorithm does neither assume the graph to be 2-connected nor needs to compute low-points (as defined in [30]) in advance. As 3-connectivity tests are used in practice and many implementation details are to consider, it would be interesting and reasonable to compare these algorithms experimentally.

Another open question is whether construction sequences provide a general new approach to test graphs efficiently on higher vertex connectivity. Up to now, no linear-time algorithm for testing graphs on 4-connectivity is known. Interestingly, construction sequences for 4-connected graphs exist [50].

References

[1] S. Albroscheit. Ein Algorithmus zur Konstruktion gegebener 3-zusammenhängender Graphen. Diploma thesis, Freie Universität Berlin, 2006.

[2] L. Auslander and S. V. Parter. On imbedding graphs in the plane. *J. Math. and Mech.*, 10(3):517–523, 1961.

[3] D. W. Barnette and B. Grünbaum. On Steinitz's theorem concerning convex 3-polytopes and on some properties of 3-connected planar graphs. In *Many Facets of Graph Theory*, pages 27–40, 1969.

[4] V. Batagelj. Inductive classes of graphs. In *Proceedings of the 6th Yugoslav Seminar on Graph Theory*, pages 43–56, Dubrovnik, 1986.

[5] V. Batagelj. An improved inductive definition of two restricted classes of triangulations of the plane. In *Combinatorics and Graph Theory, Banach Center Publications*, volume 25, pages 11–18. PWN Warsaw, 1989.

[6] P. Bertolazzi, G. Di Battista, C. Mannino, and R. Tamassia. Optimal upward planarity testing of single-source digraphs. *SIAM J. Comput.*, 27(1):132–169, 1998.

[7] M. Blum and S. Kannan. Designing programs that check their work. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC'89)*, pages 86–97, New York, 1989.

[8] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer, 2008.

[9] U. Brandes. Eager st-Ordering. In *Proceedings of the 10th European Symposium of Algorithms (ESA'02)*, pages 247–256, 2002.

[10] J. Cheriyan and K. Mehlhorn. Algorithms for dense graphs and networks. *Algorithmica*, 15(6):521–549, 1996.

[11] N. Chiba and T. Nishizeki. The Hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. *J. Algorithms*, 10(2):187–211, 1989.

[12] G. Demoucron, Y. Malgrange, and R. Pertuiset. Graphes planaires: Reconnaissance et construction de representations planaires topologiques. *Rev. Francaise Recherche Operationelle*, 8:33–47, 1964.

[13] G. Di Battista and R. Tamassia. Incremental planarity testing. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*, pages 436–441, 1989.

[14] G. Di Battista and R. Tamassia. On-line graph algorithms with *SPQR*-trees. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, pages 598–611, 1990.

[15] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *J. ACM*, 40:17–47, 1993.

[16] J. Ebert. st-Ordering the vertices of biconnected graphs. *Computing*, 30:19–33, 1983.

[17] A. Elmasry, K. Mehlhorn, and J. M. Schmidt. An O(n+m) certifying triconnnectivity algorithm for Hamiltonian graphs. *Algorithmica*, 62(3):754–766, 2012.

[18] A. Elmasry, K. Mehlhorn, and J. M. Schmidt. Every DFS tree of a 3-connected graph contains a contractible edge. *Journal of Graph Theory*, 72(1):112–121, 2013.

[19] S. Even and R. E. Tarjan. Computing an st-Numbering. *Theor. Comput. Sci.*, 2(3):339–344, 1976.

[20] S. Even and R. E. Tarjan. Corrigendum: Computing an st-Numbering (TCS 2(1976):339-344). *Theor. Comput. Sci.*, 4(1):123, 1977.

[21] D. S. Fussell, V. Ramachandran, and R. Thurimella. Finding triconnected components by local replacement. *SIAM J. Comput.*, 22(3):587–616, 1993.

[22] H. N. Gabow. Path-based depth-first search for strong and biconnected components. *Inf. Process. Lett.*, 74(3-4):107–114, 2000.

[23] Z. Galil and G. F. Italiano. Reducing edge connectivity to vertex connectivity. *SIGACT News*, 22(1):57–61, 1991.

[24] T. Gallai. Elementare Relationen bezüglich der Glieder und trennenden Punkte von Graphen. *Magyar Tud. Akad. Mat. Kutato Int. Kozl.*, 9:235–236, 1964.

[25] M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Comput.*, 5(4):704–714, 1976.

[26] A. J. Goldstein. An efficient and constructive algorithm for testing whether a graph can be embedded in a plane. Technical report, Contract No. NONR 1858-(21), Department of Mathematics, Princeton University, 1963.

[27] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In *Proceedings of the 8th International Symposium on Graph Drawing (GD'00)*, pages 77–90, 2001.

[28] C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 246–255, 2001.

[29] F. Harary and G. Prins. The block-cutpoint-tree of a graph. *Publ. Math. Debrecen*, 13:103–107, 1966.

[30] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.

[31] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[32] E. L. Johnson. A proof of the four-coloring of the edges of a regular three-degree graph. Technical report, O. R. C. 63-28 (R. R.) Min. Report, Univ. of California, Operations Research Center, 1963.

[33] D. Kratsch, R. M. McConnell, K. Mehlhorn, and J. P. Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. *SIAM J. Comput.*, 36(2):326–353, 2006.

[34] L. Lovász. Computing ears and branchings in parallel. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 464–467, 1985.

[35] E. Lucas. *Récréations Mathématiques. Part 1.* Gauthier-Villars et Fils, Paris. second edition (first edition in 1881), 1891.

[36] R. M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011.

[37] K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, 1996.

[38] K. Mehlhorn and S. Näher. From algorithms to working programs: On the use of program checking in LEDA. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS'98)*, pages 84–93, 1998.

[39] K. Mehlhorn, S. Näher, M. Seel, R. Seidel, T. Schilz, S. Schirra, and C. Uhrig. Checking geometric programs or verification of geometric structures. *Comput. Geom. Theory Appl.*, 12(1-2):85–103, 1999.

[40] K. Mehlhorn and P. Schweitzer. Progress on certifying algorithms. In *Proceedings of the 4th International Workshop on Frontiers in Algorithmics (FAW'10)*, pages 1–5, 2010.

[41] P. Mutzel. A fast $0(n)$ embedding algorithm, based on the Hopcroft-Tarjan planary test. Technical Report 107, Zentrum für Angewandte Informatik Köln, 1992.

[42] P. Mutzel. The SPQR-tree data structure in graph drawing. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP'03)*, pages 34–46, 2003.

[43] H. Nagamochi and T. Ibaraki. A linear time algorithm for computing 3-edge-connected components in a multigraph. *Japan Journal of Industrial and Applied Mathematics*, 9:163–180, 1992.

[44] A. Neumann. Implementation of Schmidt's algorithm for certifying triconnectivity testing. Master's thesis, Universität des Saarlandes and Graduate School of CS, Germany, 2011.

[45] S. Olariu and A. Y. Zomaya. A time- and cost-optimal algorithm for interlocking sets – With applications. *IEEE Trans. Parallel Distrib. Syst.*, 7(10):1009–1025, 1996.

[46] V. Ramachandran. Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity. In *Synthesis of Parallel Algorithms*, pages 275–340, 1993.

[47] J. M. Schmidt. Construction sequences and certifying 3-connectedness. In *Proceedings of the 27th Symposium on Theoretical Aspects of Computer Science (STACS'10)*, pages 633–644, 2010.

[48] J. M. Schmidt. Construction sequences and certifying 3-connectivity. *Algorithmica*, 62:192–208, 2012.

[49] R. W. Shirey. *Implementation and analysis of efficient graph planarity testing algorithms*. PhD thesis, University of Wisconsin, 1969.

[50] P. J. Slater. A classification of 4-connected graphs. *Journal of Combinatorial Theory, Series B*, 17(3):281–298, 1974.

[51] E. Steinitz. Polyeder und Raumeinteilungen. *Encyclopädie der mathematischen Wissenschaften*, 3 (Geometrie):1–139, 1922.

[52] S. Taoka, T. Watanabe, and K. Onaga. A linear time algorithm for computing all 3-edge-connected components of a multigraph. *IEICE Trans. Fundamentals E75*, 3:410–424, 1992.

[53] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[54] R. E. Tarjan. A note on finding the bridges of a graph. *Inf. Process. Lett.*, 2(6):160–161, 1974.

[55] R. E. Tarjan. Two streamlined depth-first search algorithms. *Fund. Inf.*, 9:85–94, 1986.

[56] G. Tarry. Le problème des labyrinthes. *Nouvelles Annales de Mathématique*, 14:187–190, 1895.

[57] C. Thomassen. Kuratowski's theorem. *J. Graph Theory*, 5(3):225–241, 1981.

[58] C. Thomassen. Reflections on graph theory. *J. Graph Theory*, 10(3):309–324, 2006.

[59] Y. H. Tsin. On finding an ear decomposition of an undirected graph distributively. *Inf. Process. Lett.*, 91:147–153, 2004.

[60] Y. H. Tsin. A simple 3-edge-connected component algorithm. *Theor. Comp. Sys.*, 40(2):125–142, 2007.

[61] Y. H. Tsin. Yet another optimal algorithm for 3-edge-connectivity. *J. of Discrete Algorithms*, 7(1):130–146, 2009.

[62] Y. H. Tsin and F. Y. Chin. A general program scheme for finding bridges. *Inf. Process. Lett.*, 17(5):269–272, 1983.

[63] W. T. Tutte. A theorem on planar graphs. *Trans. Amer. Math. Soc.*, 82:99–116, 1956.

[64] W. T. Tutte. A theory of 3-connected graphs. *Indag. Math.*, 23:441–455, 1961.

[65] W. T. Tutte. Connectivity in graphs. In *Mathematical Expositions*, volume 15. University of Toronto Press, 1966.

[66] W. T. Tutte. *Graph Theory*. Cambridge University Press, 1984.

[67] K.-P. Vo. Finding triconnected components of graphs. *Linear and Multilinear Algebra*, 13:143–165, 1983.

[68] K.-P. Vo. Segment graphs, depth-first cycle bases, 3-connectivity, and planarity of graphs. *Linear and Multilinear Algebra*, 13:119–141, 1983.

[69] H. Whitney. Non-separable and planar graphs. *Trans. Amer. Math. Soc.*, 34(1):339–362, 1932.

[70] S. G. Williamson. Embedding graphs in the plane — algorithmic aspects. In *Combinatorial Mathematics, Optimal Designs and Their Applications*, volume 6 of *Annals of Discrete Mathematics*, pages 349–384. Elsevier, 1980.

[71] S. G. Williamson. Depth-first search and Kuratowski subgraphs. *J. ACM*, 31(4):681–693, 1984.