Structure and Constructions of 3-Connected Graphs

Dissertation zur Erlangung des Doktorgrades

vorgelegt am

Fachbereich Mathematik und Informatik
der Freien Universität Berlin

2010

von

Jens M. Schmidt

Institut für Informatik
Freie Universität Berlin
Takustraße 9
14195 Berlin
jens.schmidt@inf.fu-berlin.de

gefördert durch die Deutsche Forschungsgemeinschaft im Rahmen
des Graduiertenkollegs *„Methods for Discrete Structures“* (GRK 1408)

Betreuer:	Prof. Dr. Günter Rote Institut für Informatik Freie Universität Berlin Takustraße 9 14195 Berlin Germany rote@inf.fu-berlin.de phone: +49 30 838 75150	
Gutachter:	Prof. Dr. Günter Rote Institut für Informatik Freie Universität Berlin Takustraße 9 14195 Berlin Germany rote@inf.fu-berlin.de phone: +49 30 838 75150	Prof. Dr. Kurt Mehlhorn Department 1: Algorithms and Complexity Max-Planck-Institut für Informatik Campus E1 4 66123 Saarbrücken Germany phone: +49 681 9325 100

Vorlage zur Begutachtung:	16. Dezember 2010
Termin der Disputation:	18. März 2011
Fassung vom:	18. März 2011

Contents

Abstract

The class of 3-connected (i. e., 3-vertex-connected) graphs has been studied intensively for many reasons in the past 50 years. One algorithmic reason is that graph problems can often be reduced to handle only 3-connected graphs; applications include problems in graph drawing, problems related to planarity and online problems on planar graphs.

It is possible to test a graph on being 3-connected in linear time. However, the linear-time algorithms known are complicated and difficult to implement. For that reason, it is essential to check implementations of these algorithms to be correct. A way to check the correctness of an algorithm for every instance is to make it *certifying*, i. e., to enhance its output by an easy-to-verify certificate of correctness for that output. However, surprisingly few work has been devoted to find certifying algorithms that test 3-connectivity; in fact, the currently fastest algorithms need quadratic time.

Two classic results in graph theory due to Barnette, Grünbaum and Tutte show that 3-connected graphs can be characterized by the existence of certain inductively defined constructions. We give new variants of these constructions, relate these to already existing ones and show how they can be exploited algorithmically. Our main result is a linear-time certifying algorithm for testing 3-connectivity, which is based on these constructions. This yields also simple certifying algorithms in linear time for 2-connectivity, 2-edge-connectivity and 3-edge-connectivity. We conclude this thesis by a structural result that shows that one of the constructions is preserved when being applied to depth-first trees of the graph only.

Zusammenfassung

Die Klasse der 3-zusammenhängenden Graphen ist aus vielen Gründen seit 50 Jahren Gegenstand aktiver Forschung. Ein algorithmisch geprägter Grund ist, dass viele Graphenprobleme auf 3-zusammenhängende Graphen reduziert werden können. Anwendungen finden sich beispielsweise bei Graphzeichnungsproblemen und Problemen auf planaren Graphen.

Ein Graph kann in Linearzeit auf 3-Zusammenhang überprüft werden. Allerdings sind die bekannten Linearzeitalgorithmen kompliziert und schwierig zu implementieren. Aus diesem Grund ist es unabdingbar, Implementationen dieser Algorithmen auf Korrektheit zu überprüfen. Eine Möglichkeit, die Korrektheit jeder Probleminstanz zu überprüfen, ist, den Algorithmus *zertifizierend* zu machen. Ein zertifizierender Algorithmus ist ein Algorithmus, der neben der Ausgabe zusätzlich ein leicht zu verifizierendes Korrektheitszertifikat dieser Ausgabe liefert. Trotz zahlreicher Anwendungen 3-zusammenhängender Graphen wurden zertifizierende Algorithmen, die 3-Zusammenhang testen, bisher kaum untersucht; die schnellsten bekannten Algorithmen benötigen quadratische Laufzeit.

Zwei klassische Resultate aus der Graphentheorie von Barnette, Grünbaum und Tutte charakterisieren 3-zusammenhängende Graphen durch die Existenz induktiv definierter Konstruktionen. Wir untersuchen neue Varianten dieser Konstruktionen, stellen diese zu den klassischen Resultaten in Beziehung und zeigen, wie die Konstruktionen algorithmisch genutzt werden können. Der Hauptbeitrag dieser Arbeit ist ein auf diesen Konstruktionen basierender, zertifizierender Algorithmus, der Graphen auf 3-Zusammenhang in Linearzeit testet. Dieser kann auf k-Zusammenhang und k-Kantenzusammenhang für $k \leq 3$ erweitert werden. Abschließend zeigen wir als Strukturresultat, dass eine der vorgestellten Konstruktionen erhalten bleibt, wenn sie nur auf einen Teil des Graphen, nämlich einen Tiefensuchbaum, angewendet wird.

Acknowledgments

I would like to thank all the people that have supported me in writing this thesis. First of all, I thank my advisor, Günter Rote, for many good ideas, for being always patient and understanding, for his support and for the chance to work on the best dissertation topic that I can imagine. I would also like to thank Helmut Alt for his support, in particular, for giving me the opportunity to teach during the last months.

I am grateful to my colleagues from the work group "Theoretical Computer Science" at Freie Universität Berlin and the research training group "Methods for Discrete Structures" for many interesting discussions, not only in the coffee breaks, and the excellent research environment. The latter would not have been possible without the financial support from the Deutsche Forschungsgemeinschaft.

Special thanks to Britta and Rik for being most pleasant and supporting room mates, Lena for drawing me many nice pictures on my board, Romain for great discussions about logic and a view on mathematics that is much more fundamental than the one I am used to, Hans for being always there to discuss new ideas, Claudia for giving me advice how to teach best, Astrid for very helpful organizational tips and tricks and Frank for being always the first at the printer.

This thesis would not be this thesis if I had not met Kurt Mehlhorn and Amr Elmasry from MPII Saarbrücken. I would like to thank both for intensive research on every aspect of 3-connectivity, many interesting and fruitful discussions and, last but not least, for their expert knowledge of hunting foxes.

Finally and most importantly, I am grateful to my whole family and to Marie for their faith in me and a support that was always constant, even in times in which I spent almost every evening at work.

Berlin, December 2010

Chapter 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. One algorithmic reason is that graph problems can often be reduced to handle 3-connected graphs; applications include problems in graph drawing (see [51] for a survey), problems related to planarity [9, 28] and online problems on planar graphs (see [8] 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 [24] but becomes solvable in linear running time [13] for higher connectivity, as 4-connected planar graphs have been proven to be Hamiltonian [75].

Outside the algorithmic world, the interest in planar 3-connected graphs stems mainly from the fact that they are precisely the graphs that form the 1-skeletons of 3-dimensional convex polytopes [61] and that they admit a unique embedding in the plane (up to flipping) [82]. Also, some of the most intriguing and long-standing open problems in graph theory as the cycle double cover conjecture [59, 62] and Barnette's conjecture [4] can be reduced to 3-connected graphs.

We want to design efficient algorithms from *inductively defined constructions* of graph classes. 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 on 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 [6, 7, 36]. Most of these constructions have not been exploited computationally, although this might give new algorithms for the corresponding recognition problems.

For an inductively defined construction of $\mathcal{C}$ and a graph $G \in \mathcal{C}$, we call the 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 *top-down* variants of construction sequences, as opposed to *bottom-up* variants.

For the class of 3-connected graphs, one of the most noted constructions is due to Tutte [76], based on the following fact: Every 3-connected graph G on more than 4 vertices contains a *contractible* edge, i. e., an edge that preserves the graph to be 3-connected upon contraction. Iteratively contracting such an edge yields 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. Unfortunately, also non-3-connected graphs can contain contractible edges, but adding a side 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 (see Section 3.1.1 or [17, 57]); we will call this a *sequence of contractions*.

It is also possible to describe the construction sequence 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 [76].

Barnette and Grünbaum [5] and Tutte (implicitly shown in Theorems 12.64 and 12.65 of [77]) give a different construction of 3-connected graphs that is based on the following argument: Every 3-connected graph G on more than 4 vertices contains a *removable* edge. *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* and *removable* edges and show how to add a side condition to fully characterize 3-connected graphs in Section 3.2.2. Again, the original 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 [68, 69, 77], 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 [48, 57] that both compute a sequence of contractions. The latter is part of this thesis and is given in Section 3.5. This algorithm can additionally compute a construction sequence that uses only removals in $O(|V|^2)$. One of its building blocks is a straight-forward transformation from the sequence of removals to the sequence of contractions in time $O(|E|)$ (see Section 3.2.3). It is important to note that all mentioned algorithms do not rely on the 3-connectivity test of Hopcroft and Tarjan [35], which runs in linear time but is rather involved. Until now, we are not aware of any algorithm that computes any of these sequences in subquadratic time.

We will show in Chapter 4 how to improve the quadratic-time algorithm to linear time. One of the main contributions is therefore an optimal 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 the paths of a simple decomposition such that these groups of paths can be decomposed later into the desired operations for Barnette's and Grünbaum's construction. This has a number of consequences.

Top-down and bottom-up variants of both constructions. One can immediately obtain the sequence of removals from Barnette's and Grünbaum's bottom-up construction sequence by replacing every operation with its inverse removal operation. Applying the transformation of Section 3.2.3 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 [10] initiated the concept of programs that check their work. Mehlhorn and Näher [39, 46, 47] (see [44] for an extensive survey) introduced the concept of *certifying algorithms*, which give an easy-to-verify certificate of correctness along with their output. Achieving certifying algorithms is a major goal for problems where the fastest solutions known are complicated and difficult to implement. Testing a graph on 3-connectivity is such a problem, but surprisingly few work has been devoted to certify 3-connectivity, although sophisticated linear-time recognition algorithms (not giving an easy-to-verify certificate) are known for over 35 years [35, 79, 80].

The currently fastest algorithms that certify 3-connectivity need $O(|V|^2)$ time and use construction sequences as certificates [1, 48, 57] (see Section 3.5 for the latter algorithm). Recently, based on the results in Chapter 5 of this thesis, 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 [44, Chapter 5.4, p. 26] [17].

We solve this problem by giving a linear-time certifying algorithm for 3-connectivity that uses Barnette's and Grünbaum's construction sequence as certificate. The certificate can be easily verified in time $O(|E|)$, as shown in Section 3.6.2. This implies a new, path-based and certifying test on 3-connectivity in linear time and space, which is simple-to-implement and uses two passes over the graph (the first one being a depth-first search).

In contrast to the non-certifying algorithms in [35, 79, 80], the test does neither assume the graph to be 2-connected nor needs to compute low-points (see [35] for a definition of low-points); instead, it uses the structure of 3-connected graphs implicitly by applying simple path-generating rules. We also give a linear-time decomposition of graphs that is closely related to ear decompositions [42, 83] in Section 4.1, which unifies existing tests on 2-connectivity [12, 16, 18, 19, 21, 64, 66] and 2-edge-connectivity [65, 71, 74].

Certifying 3-edge-connectivity in linear time. We are not aware of any 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 [22, 52, 63, 72, 73]. The first (non-certifying) one due to Galil and Italiano [22] uses the fact that testing a graph G on k-edge-connectivity can be reduced to test 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 (see Section 3.6.3).

We additionally strengthen Tutte's result that every 3-connected graph G on more than 4 vertices contains a contractible edge; Chapter 5 shows that every depth-first search tree of G contains a contractible edge. Generalizing this statement to spanning trees instead of depth-first search trees is not possible, as counterexamples show. However, we prove that if G is 3-regular or does not contain two disjoint pairs of adjacent degree-3 vertices, every spanning tree of G contains a contractible edge.

Preliminaries for this thesis are given in Chapter 2. We recapitulate old and develop new results about construction sequences in Chapter 3, including a computational approach in $O(n^2)$ time. In addition, easy-to-verify certificates are discussed. Using these results, Chapter 4 describes a linear-time certifying algorithm for 3-connectivity, which yields also linear-time certifying algorithms for 2-connectivity and k-edge-connectivity with $k \leq 3$. Chapter 5 gives structural results about contractible edges in spanning trees of 3-connected graphs.

Chapter 2

Preliminaries

A *graph* G is an ordered pair $(V(G), E(G))$ that consists of a set $V(G)$ of *vertices* and a set $E(G)$ of *edges* that is disjoint from $V(G)$, together with a mapping that maps each edge to a pair of not necessarily distinct vertices. Let a graph G be *finite* if both $V(G)$ and $E(G)$ are finite. A graph G is *directed* if every edge is mapped to an ordered vertex pair and *undirected* if every edge is mapped to an unordered vertex pair. We consider only graphs that are finite and undirected.

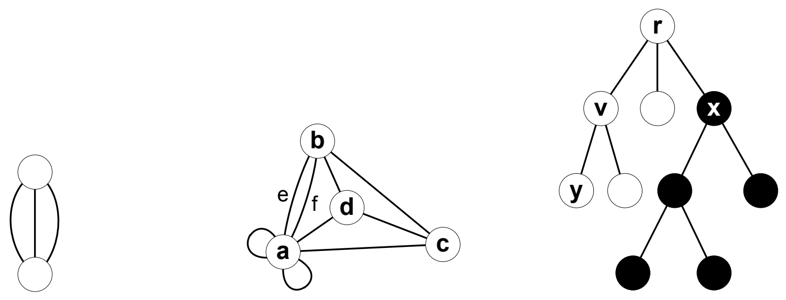
For a graph G, we denote the number of its vertices by $n = |V(G)|$ and the number of its edges by $m = |E(G)|$. For an edge e, the elements of the vertex pair to which e is mapped are the *end vertices* of e. For convenience, we denote an edge with end vertices v and w by vw. Let an edge $e = vw$ in a graph G be a *self-loop* if $v = w$ and let e be *parallel* if e is no self-loop and G contains an edge with end vertices v and w different from e. A graph is *simple* if it contains neither a self-loop nor a parallel edge. The *underlying simple graph* of a graph G is the graph generated from G by deleting all self-loops and replacing, for every two vertices v and w, the set A of all edges vw in G by the edge vw if $A \neq \emptyset$.

Two graphs G and H are *isomorphic* if there are bijections $\phi : V(G) \to V(H)$ and $\theta : E(G) \to E(H)$ such that $e = vw$ is an edge in G if and only if $\theta(e) = \phi(v)\phi(w)$ is an edge in H. Note that, for simple graphs, the bijection θ is completely determined by ϕ and can therefore be omitted in the definition. Unless stated otherwise, we will not distinguish between two isomorphic graphs G and H, i. e., we will write $G = H$ in these cases.

Two vertices v and w in a graph G are *adjacent vertices* (or, equivalently, *neighbors*) if G contains an edge vw. A vertex v is *incident* to an edge e if v is an end vertex of e. A graph G is *complete* if every two distinct vertices in G are adjacent. Let a K_n*-multigraph* be a complete graph on n vertices and let K_n be the *complete simple graph* on n vertices. The graph K_0 is also called the *empty graph*, as it does not contain any vertex and, thus, does not contain any edge. Let K_2^m be the K_2-multigraph that contains exactly m parallel edges and no self-loop.

Let the *degree* of a vertex v in a graph G, denoted by $deg(v)$, be the number of edges that are incident with v, each self-loop counting as two edges. For a vertex v in a graph G, let $N(v) = \{w \mid vw \in E(G)\}$ denote its *set of neighbors* in G. Note that in simple graphs $deg(v) = |N(v)|$ for every vertex v. For a graph $G \neq K_0$, let $\delta(G)$ be the *minimum degree* of all vertices in G; for the empty graph K_0, we define $\delta(K_0) = 0$. A graph G is *k-regular* if every vertex in G has degree k. Graphs that are 3-regular are also called *cubic*.

For a graph G, a graph H with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ is called a *subgraph* of G, denoted by $H \subseteq G$. Note that $V(H)$ is not an arbitrary subset of $V(G)$, since defining H as graph implies that $V(H)$ contains the end vertices of every edge in $E(H)$. For $H \subseteq G$, we also say that G *contains* H. If $H \subseteq G$ and $H \neq G$, H is called a *proper subgraph*, denoted by $H \subset G$. For a given graph class C, a graph $G \in C$ is *minimal* (respectively, *maximal*) with respect to some property if C does not contain another graph H with that property such that $H \subset G$ (respectively, $G \subset H$). A subgraph H of a graph G is *spanning* if $V(H) = V(G)$. A *decomposition* of a graph G is a set of subgraphs of G whose edge-sets partition $E(G)$.



(a) K_2^3, the unique graph (up to isomorphisms) on two vertices with three parallel edges and no self-loop.

(b) A K_4-multigraph G with $n = 4$, $m = 9$, two self-loops, two parallel edges, $deg(a) = 8$, $N(a) = V(G)$ and $\delta(G) = 3$. As G is complete, G has no vertex cut; it however has edge-cuts, e. g., $\{ca, cb, cd\}$.

(c) A tree T with root r and a child x of r. $T(x)$ is the subtree of T that consists of black vertices. A traversal of the (unique) path $x \rightarrow_T y$ visits the vertices x, r, v and y in that order.

Figure 2.1: Three undirected and connected graphs.

We define fundamental operations on graphs. Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. Whenever two or more graphs are mentioned, we follow the convention that the union of all vertex sets and the union of all edge sets is disjoint, i. e., in this case $(V \cup V') \cap (E \cup E') = \emptyset$. This ensures in particular that the following union-operation generates a graph. We define $G \cup G' = (V \cup V', E \cup E')$ and $G \cap G' = (V \cap V', E \cap E')$.

For any edge e and a graph G, *deleting* e is the operation of subtracting e from

$E(G)$. Conversely, *adding* e adds e to $E(G)$. For any vertex v and a graph G, *deleting* v is the operation of deleting every edge in G that is incident to v followed by subtracting v from $V(G)$; *adding* v adds v to $V(G)$. Note that deleting a vertex without deleting its incident edges does not always yield a graph. For a graph G and any set X of either vertices or edges, let $G \setminus X$ denote the graph that is generated from G by deleting all elements of X in G. If X consists of just one element x, we also write $G \setminus x$ instead of $G \setminus \{x\}$. *Identifying* two vertices x and y in a graph G deletes y and adds for every edge yz that was incident to y in G an edge xz if $y \neq z$ and an edge xx, otherwise.

A *path* P is a graph with vertex set $V(P) = \{v_0, v_1 \dots, v_k\}$ and the edge set $E(P) = \{v_0v_1, v_1v_2, \dots, v_{k-1}v_k\}$ such that $k \geq 0$ (the edge set is empty if $k = 0$). Let a vertex v_i in P be an *end vertex* of P if $v_i \in \{v_0, v_k\}$ and an *inner vertex* of P otherwise. Although a path is an undirected graph, we will often impose a direction on it, e. g., in order to state the direction in which the path is traversed in an algorithm. Let $v \rightarrow_G w$ denote a path from vertex v to vertex w that is contained in a graph G and let $s(P) = v$ and $t(P) = w$ be the source vertex and the target vertex of P, respectively. Two or more paths are *vertex-disjoint* if their vertex sets are pairwise disjoint. Two or more paths are *internally vertex-disjoint* if none of them contains an inner vertex of another.

A graph G is *connected* if there is a path between every two vertices in G and *disconnected* otherwise. The *connected components* of a graph G are its maximal connected subgraphs. Note that a graph has more than one connected component if and only if it is disconnected. A *cycle* is a simple 2-regular connected graph. The *length* of paths and cycles is the number of edges they consist of. A cycle that is a spanning subgraph of a graph G is called a *Hamiltonian cycle* of G.

A cycle with an additional vertex v that is adjacent to all cycle vertices is called a *wheel* graph; the edges incident to v are called *spokes*. Let W_n be the wheel graph on n vertices, i. e., the wheel graph with $n - 1$ spokes. A graph is *acyclic* if it does not contain a cycle. A simple acyclic graph is a *forest*. A connected forest is called a *tree*. Let a *subtree* of a tree T be a connected subgraph of T. Note that a subtree is again a tree. A tree T is *rooted* if $n = 0$ or one vertex of T is specified as the *root*.

For two not necessarily distinct vertices v and w in a tree T that is rooted at r, let v be an *ancestor* of w and w be a *descendant* of v if $v \in V(w \rightarrow_T r)$. If additionally $v \neq w$ holds, v is called a *proper* ancestor and w is called a *proper* descendant. For two vertices v and w in a rooted tree T, let v be the *parent* of w and w be a *child* of v if v is an ancestor of w and the length of $v \rightarrow_T w$ is one. A vertex in a rooted tree that has no child is called *leaf*.

Let T be a tree with root r, let v be a vertex in T different from r and let A be the connected component of $T \setminus v$ that contains r. We define $T(v) = T \setminus V(A)$, i. e., $T(v)$ is the maximal rooted subtree of T with root v. Let F be a forest that consists of rooted trees. Let a strict total order $\prec$ on $V(F)$ be a *pre-order* if, for every vertex $v \in V(F)$, all proper ancestors of v in F precede v in $\prec$.

A *vertex cut* in a graph G is a subset X of $V(G)$ such that $G \setminus X$ is disconnected. Similarly, an *edge cut* in a graph G is a subset X of $E(G)$ such that $G \setminus X$ is disconnected. Vertex cuts of size one, two and three are called *cut vertices*, *separation pairs* and *separation triples*, respectively. Edge cuts of size one are called *bridges*. Note that $\emptyset$ is a vertex and edge cut in a graph if and only if the graph is disconnected. The complete graphs are the only graphs that do not have vertex cuts, whereas the complete graphs with $n \leq 1$ (i. e., K_0 and the K_1-multigraphs) are the only graphs that do not have edge cuts.

2.1 Notions of Vertex-Connectivity

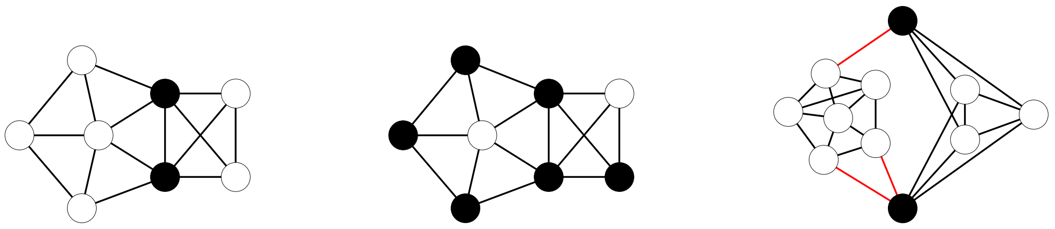
The notion of *k-connectivity* (sometimes referred to as *k-vertex-connectivity*, *k-connectedness* or *k-tuply-connectivity*) is used inconsistently in literature, in particular for complete graphs and for small graphs. For three representative definitions that are not equivalent, but well-established, see the excellent text books of Bondy and Murty [11, p. 207], Diestel [15, p. 11] and Tutte [78, p. 70]. We first give a short comparison of these definitions and then state which one we will use.

The definition of k-connectivity in [11] is based on internally vertex-disjoint paths, whereas the one in [15] is based on the existence of special vertex cuts, as introduced by Whitney [82]. For a graph G, the given definitions differ if and only if $n < 2$ or G is a complete graph that contains at least two parallel edges vw for every two vertices v and w. The reason is that, besides different conventions for small graphs, additional parallel edges can increase the number of internally vertex-disjoint paths between each two vertices for complete graphs. Although this leads to a higher vertex-connectivity of some complete graphs for the definition of [11], the vertex cuts of G are preserved (there is none). If desired, both definitions can be made equivalent for general graphs by considering complete graphs separately. For simple graphs on at least two vertices, both definitions are equivalent.

The definition of k-connectivity due to Tutte [78] is conceptually different from the first two, as it is motivated by its generalization to matroids and the attempt to remain invariant under taking dual graphs (see also [56, pp. 271]). It characterizes k-connectivity not only by the size of vertex cuts but also by the number of edges in each of the two disjoint subgraphs induced by such vertex cuts. This leads to a considerably different characterization of k-connectivity in general. Contrary to the first two definitions, deleting edges can increase the vertex-connectivity of a graph. However, for simple connected graphs with $n > 2$ and $k \in \{2, 3\}$, all three definitions of k-connectivity are equivalent [78, Theorem IV.10, p. 74].

Choosing the right definition is dependent on the application. If complete graphs occur and parallel edges should make a difference in vertex-connectivity, the definition based on internally vertex-disjoint paths may be used. If only simple graphs are considered or the vertex-connectivity should be independent on parallel edges and self-loops, the definition based on vertex cuts may be used. Finally, if a generalization to matroids is important or the vertex-connectivity should be invariant under taking dual graphs, the definition in [78, p. 70] is appropriate.

We use the definition that is based on vertex cuts, as, for our purposes, vertex-connectivity should be independent from self-loops and parallel edges. This is no limitation; we can adapt all algorithms and theorems given in this thesis to the other definitions as follows. The above comparison lists the graphs for which the given definitions of k-connectivity may differ for $k \leq 3$ (we do not deal with higher vertex-connectivity). These graphs are the non-simple graphs and the complete graphs. Both can be detected along with their edge multiplicities in time $O(n+m)$ by applying two bucket sorts on the end vertices of edges in $E(G)$. For each definition, the k-connectivity of such graphs for $k \leq 3$ is only dependent on the multiplicity of parallel edges and self-loops. This preserves the (at least linear) asymptotic running time of all algorithms in this thesis when a different definition of k-connectivity is used.



(a) A graph G with $\kappa(G) = 2$, as G contains no cut vertex but the separation pair depicted with black vertices. Moreover, $\kappa'(G) = \delta(G) = 3$.

(b) A vertex cut in G of size $n-2$ that contains the separation pair of Figure (a).

(c) A graph H with $\kappa(H) = 2$, $\kappa'(H) = 3$ (as the red edge cut is of minimal size 3) and $\delta(H) = 4$.

Figure 2.2: Examples of graphs that are not 3-connected.

We define every graph to be 0-*connected* and 0-*edge-connected*. For $k \geq 1$, let a graph G be k-*connected* if $n > k$ and there is no vertex cut X in G with $|X| < k$. A graph is therefore 1-connected if and only if it is connected and $n > 1$. Note that k-connectivity does neither depend on parallel edges nor on self-loops. Complete graphs have no vertex cuts and so their vertex-connectivity is only limited by the side condition $k < n$, implying that they are $(n-1)$-connected, but not n-connected. The side condition $k < n$ allows also to replace the condition $|X| < k$ by $|X| = k-1$ in the definition of vertex-connectivity, as every vertex cut X of size $|X| \leq k-2 < n-2$ can be augmented to contain $n-2$ vertices (see Figure 2.2(b)).

Let the *connectivity* $\kappa(G)$ of a graph G be the largest integer k for which G is k-connected. Thus, for $n \geq 1$, $\kappa(G) = n-1$ if and only if G is a non-empty

K_n-multigraph and $\kappa(G) = 0$ if and only if $n \leq 1$ or G is disconnected.

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 the *edge-connectivity* $\kappa'(G)$ of a graph G be the largest integer k for which G is k-edge-connected. The following fundamental lemma links the *connectivity*, *edge-connectivity* and *minimum degree* of every graph (see also Harary [31] for a proof in modern graph theory notation).

Lemma 1 (Whitney [82])**.** *For every graph G holds $\kappa(G) \leq \kappa'(G) \leq \delta(G)$.*

Lemma 1 implies that every k-connected graph is k-edge-connected and has minimum degree k. Note that K_1 does not contain an edge and therefore must have connectivity and edge-connectivity 0. This is the motivation for defining K_1-multigraphs as 0-connected and 0-edge-connected, although they are connected.

The following theorem characterizes vertex-connectivity in terms of internally vertex-disjoint paths and is usually ascribed to Menger. Many different variants exist [58]. We state a variant for (global) vertex-connectivity in undirected graphs due to Whitney.

Theorem 2 (Menger [49], in the variant of Whitney [82])**.** *A graph G is k-connected for $k \geq 1$ if and only if $n > k$ and G contains at least k internally vertex-disjoint paths between every two vertices.*

The following three results are well-known corollaries of Theorem 2 and will be used throughout this thesis.

Lemma 3 (Fan Lemma [11, Proposition 9.5])**.** *Let v be a vertex in a graph G that is k-connected and let A be a set of at least k vertices in G with $v \notin A$. Then there are k internally vertex-disjoint paths $P_1, \ldots, P_k$ from v to distinct vertices $a_1, \ldots, a_k \in A$ such that $V(P_i) \cap A = a_i$ for each $1 \leq i \leq k$.*

Lemma 4 ([11, Proposition 9.5])**.** *Let A and B be two sets of at least k vertices in a k-connected graph G. Then there are k vertex-disjoint paths $P_1, \ldots, P_k$ in G with $s(P_i) \in A$ and $t(P_i) \in B$ for each $1 \leq i \leq k$.*

Lemma 5 (Expansion Lemma, Lemma 4.2.3 in [81])**.** *Let G be a k-connected graph. Then the graph that is generated from G by adding a new vertex v and adding edges such that v is adjacent to at least k vertices in $V(G)$ is k-connected.*

Chapter 3

Construction Sequences

...you first have to divide the graph into pieces.

Robert E. Tarjan [20]

We recapitulate old and develop new results about construction sequences. We show how these results can be used to compute the sequence of contractions and the sequence of removals in quadratic time. Based on these sequences, certificates for the 3-connectivity and 3-edge-connectivity of graphs are given that can be easily verified.

3.1 Tutte's Construction Sequence

The operation of *contracting* an edge $e = xy$, $x \neq y$, in a graph deletes e and identifies (merges) x and y. Let an edge $e = xy$, $x \neq y$, in a graph be *contractible* if contracting e generates a 3-connected graph (see Figure 3.1). Note that contractions in a graph G can generate parallel edges and self-loops, even if G is simple.

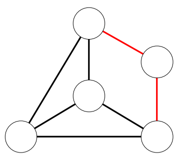


Figure 3.1: A graph that contains exactly two contractible edges, which are depicted in red.

The operation of *vertex splitting* in a graph G replaces a vertex $v \in V(G)$ of degree at least 4 (not counting self-loops) by two vertices x and y, inserts the edge xy and replaces every former edge uv in G, $u \neq v$, with either the edge ux or uy such that $|N(x)| \geq 3$ and $|N(y)| \geq 3$ in the new graph (not counting self-loops). In addition, every self-loop in G that is incident to v is replaced by either the parallel edge xy or one of the self-loops xx and yy. As proven by Tutte, vertex splittings preserve 3-connectivity.

Lemma 6 (Tutte [76])**.** *Applying a vertex splitting on a* 3*-connected graph generates a* 3*-connected graph.*

It is easy to see that every vertex splitting can be inversed by contracting the edge xy between the two new vertices x and y.

Lemma 7. *Every vertex splitting on a* 3*-connected graph G can be inversed by contracting a contractible edge.*

Every wheel graph is 3-connected. In particular, $K_4 = W_4$ is the unique smallest 3-connected graph (with respect to the number of edges). A variant of the famous *Wheel Theorem* of Tutte [76] uses wheel graphs to give the following characterization of simple 3-connected graphs.

Theorem 8 (Tutte [76], [26, Theorem 7.19, p. 152])**.** *A simple graph is* 3*-connected if and only if it can be generated from a wheel graph by repeatedly adding edges between non-adjacent vertices and applying vertex splittings.*

We can easily extend Theorem 8 to deal with non-simple graphs G by allowing to add parallel edges and self-loops. Moreover, at the expense of using parallel edges in intermediate graphs, every wheel graph on more than 4 vertices can be generated from a K_4-multigraph by vertex splittings. This allows us to restrict the construction sequence of Theorem 8 (in the variant allowing parallel edges and self-loops) to use only one operation, namely vertex splitting, by taking K_4-multigraphs as base graphs instead of wheel graphs.

All edge-additions of the former construction sequence can be replaced by appropriate parallel edges and self-loops in the K_4-multigraph (see Figure 3.2 for an example how these edges can be constructed). We obtain the following theorem.

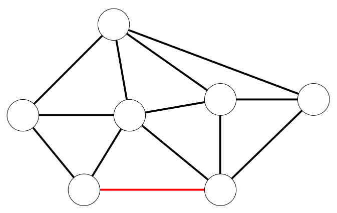
Theorem 9. *(Corollary of Theorem 8) A graph G is* 3*-connected if and only if there is a construction sequence from a K_4-multigraph to G using vertex splittings.*

Theorem 9 implies in particular the following classic result due to Tutte when combined with Lemma 7.

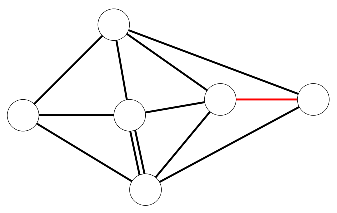
Corollary 10 (Tutte [76])**.** Every 3-connected graph on more than 4 vertices contains a contractible edge.

3.1.1 Contractions

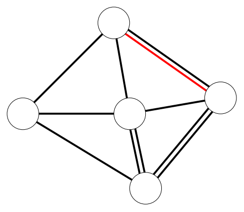
According to Lemma 7, every vertex splitting on a 3-connected graph can be inversed by contracting a contractible edge. The converse is not true in general, as Figure 3.3 exemplifies. We show in which cases the inverse operation of contracting a contractible edge is a vertex splitting.



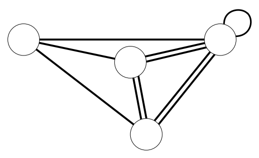
(a) A 3-connected graph G containing a red contractible edge.



(b) Contracting the red contractible edge in (b) generates a non-simple but 3-connected graph.



(c) The result of contracting the red contractible edge in Figure (b).



(d) The K_4-multigraph after contracting the red contractible edge in Figure (c).

Figure 3.2: Contractions and vertex splittings. Figures (a)–(d) depict a sequence of contractions from a 3-connected graph G to a K_4-multigraph on contractible edges whose end vertices have both at least 3 neighbors. Each contraction can be inversed by a vertex splitting.

Lemma 11. *The inverse operation O of contracting a contractible edge $e = xy$ in a graph G is a vertex splitting if and only if $|N(x)| \geq 3$ and $|N(y)| \geq 3$ (not counting self-loops).*

Proof. Let G' be the 3-connected graph that is generated from G by contracting e and let v be the vertex in G' to which e is contracted. If O is a vertex splitting, G is 3-connected with Lemma 6 and $|N(x)| \geq 3$ and $|N(y)| \geq 3$ follows.

If O is not a vertex splitting, assume to the contrary that $|N(x)| \geq 3$ and $|N(y)| \geq 3$ in G (we do not count self-loops for degrees and neighbors in this proof). Then, according to the definition of vertex splittings, $deg(v) < 4$ must hold, as otherwise O would be a vertex splitting. Since G' is 3-connected, v has at least 3 neighbors in G', which implies $deg(v) = 3$. However, due to the contraction operation, the degree of v is the number of the edges in G that are incident to exactly one of x and y. Because of $|N(x)| \geq 3$ and $|N(y)| \geq 3$, there are at least 4 such edges in G, which contradicts $deg(v) = 3$. □

Combining Lemma 11 with Lemma 7 and Theorem 9 gives the following result.

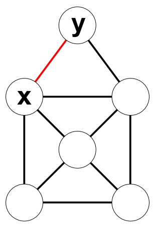


Figure 3.3: The edge xy in this graph G is contractible, since contracting xy results in a wheel graph with an additional parallel edge. However, the inverse operation of contracting xy is not a vertex splitting, because $|N(y)| = 2$ in G.

Theorem 12 (see also [17, 57])**.** *A graph G is 3-connected if and only if there is a sequence of contractions from G to a K_4-multigraph on contractible edges $e = xy$ with $|N(x)| \geq 3$ and $|N(y)| \geq 3$.*

For 3-connected graphs, we call the sequences of Theorems 9 and 12 *Tutte's construction sequence* in the *bottom-up* variant and the *top-down* variant, respectively. The top-down variant is also called a *sequence of contractions*. We remark that both construction sequences can be transformed efficiently to each other (in linear time for suitable representations of the construction sequences) by simply replacing vertex splittings with contractions and vice versa.

We show that the equivalence of Theorem 12 still holds when the condition that contracted edges have to be contractible is omitted.

Theorem 13. *A graph G is 3-connected if and only if there is a sequence of contractions from G to a K_4-multigraph on edges $e = xy$ with $|N(x)| \geq 3$ and $|N(y)| \geq 3$.*

Proof. If G is 3-connected, the claimed construction sequence exists due to Theorem 12.

Otherwise, let Q be a sequence of contractions from G to a K_4-multigraph on edges $e = xy$ with $|N(x)| \geq 3$ and $|N(y)| \geq 3$. Assume to the contrary that G is not 3-connected. Then, as every K_4-multigraph is 3-connected, there is a contraction in Q that is applied on a non-3-connected graph G_{i+1} and generates a 3-connected graph G_i. Let $e = xy$ be the edge in G_{i+1} that is contracted by Q. Then e is contractible and satisfies $|N(x)| \geq 3$ and $|N(y)| \geq 3$ in G_{i+1}. According to Lemma 11, the inverse operation of contracting e is a vertex splitting. This implies with Lemma 6 that G_{i+1} is 3-connected, which contradicts the choice of G_{i+1}. □

3.1.2 Computation in Quadratic Time

We describe next a straight-forward $O(n^2)$ algorithm that computes a sequence of contractions for an input graph G in the case when G is 3-connected. The algorithm uses the linear-time test on 3-connectivity of Hopcroft and Tarjan [35]; for a quadratic-time algorithm that does not use this test see [48]. We can assume that

G contains more than 4 vertices, as otherwise $|V(G)| = 4$ holds and the sequence is empty. We first decrease the number of edges in G to $O(n)$ by applying the following algorithm due to Nagamochi and Ibaraki.

Theorem 14 (Nagamochi and Ibaraki [53]). *Let G be a connected graph and $k \in \mathbb{N}$. Then there is an $O(m)$ time algorithm that computes a spanning subgraph of G with at most $k(n-1)$ edges such that this spanning subgraph is k-connected (resp. k-edge-connected) if and only if G is k-connected (resp. k-edge-connected). Moreover, if G is k-connected (resp. k-edge-connected), the spanning subgraph contains a vertex of degree k.*

Applying this algorithm on a graph G preserves it to be 3-connected, or respectively, to be not 3-connected. If the input graph is 3-connected, the resulting graph contains a vertex of degree 3. By a result of Halin [30], every vertex v of degree 3 in a 3-connected graph on more than 4 vertices is incident to a contractible edge e.

We can compute e in time $O(n)$ by contracting each of the three edges that are incident to v and testing each generated graph with the algorithm of Hopcroft and Tarjan on 3-connectivity. After contracting e, repeatedly applying both steps in time $O(n)$, the one of Nagamochi-Ibaraki and the contracting step, generates the whole sequence of contractions in $O(n^2)$ time. Note that edges may be omitted in every step due to the algorithm of Nagamochi and Ibaraki. By applying the sequence of contractions to G, these edges will result in additional parallel edges and self-loops in the K_4-multigraph. This shows that also the generated K_4-multigraph can be computed in time $O(m)$ if needed.

We will give a more general approach for computing this and several other construction sequences in Section 3.5.

3.2 Barnette's and Grünbaum's Construction Sequence

A *subdivision* of a graph G is a graph that is generated from G by replacing each edge in $E(G)$ 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 a graph G be the operation that, if $deg(v) = 2$ and $|N(v)| = 2$ (both not counting self-loops), deletes v followed by adding an edge between its neighbors. For a graph G, let $smooth(G)$ be the graph that is generated from G by smoothing every vertex.

We define the operations of Barnette and Grünbaum.

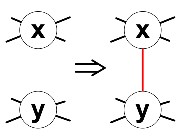
Definition 15. An operation on a graph is called a *Barnette-Grünbaum-operation* (*BG-operation*) if it is one of the following three operations (see Figures 3.4(a)–(c)).

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

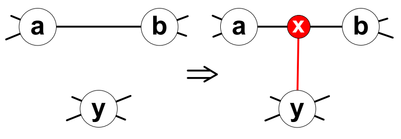
(b) subdivide an edge ab that is not a self-loop by a vertex x and add the edge xy for a vertex $y \notin \{a, b\}$

(c) for two distinct and non-parallel edges e and f that are not self-loops, subdivide e by the vertex x, subdivide f by the vertex y and add the edge xy

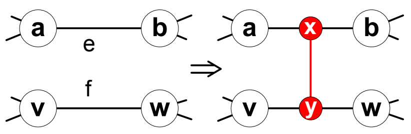
In all three cases, let xy be the edge that was *added* by the BG-operation.



(a) The added edge xy may be a parallel edge.



(b) The vertices y, a and b are pairwise distinct.



(c) The edges e and f are neither identical nor parallel. However, they may have one common end vertex.

Figure 3.4: The three BG-operations.

BG-operations do not only preserve 3-connectivity when applied to a 3-connected graph, they also lead to a complete characterization of the class of 3-connected graphs. The following classical results were proven in this notation by Barnette and Grünbaum [5], but also described implicitly in theorems of Tutte about *nodal connectivity* [77, Theorems 12.64 and 12.65] and *constructions of* 3*-connected graphs* [78, Theorems IV.14 – IV.18].

Lemma 16 (Barnette and Grünbaum [5], Theorems IV.14 – IV.18 in Tutte [78])**.** *Applying a BG-operation on a* 3*-connected graph generates a* 3*-connected graph.*

Theorem 17 (Barnette and Grünbaum [5], Theorems 12.64 and 12.65 in Tutte [77])**.** *A graph G without self-loops is* 3*-connected if and only if G can be constructed from K_4 using BG-operations.*

For 3-connected graphs without self-loops, we call the sequence of Theorem 17 *Barnette's and Grünbaum's construction sequence* (in the *bottom-up* variant) or a *construction sequence using BG-operations*. As an immediate consequence of Lemma 16, all intermediate graphs in a construction sequence using BG-operations are 3-connected.

Let a k-connected graph G be *minimally* k-connected if the deletion of every edge in G generates a graph that is not k-connected. Theorem 17 implies that the last BG-operation to construct a minimally 3-connected graph is either operation 15b or 15c. This gives the following corollary, which is also a corollary of Halin's work in [29].

Corollary 18 (Halin [29])**.** Every minimally 3-connected graph contains a vertex of degree 3.

3.2.1 Basic Sequences

Let an operation that is applied on a graph be *basic* if it creates neither a new parallel edge nor a new self-loop. Let a construction sequence be *basic* if it uses only basic operations. Of course, we cannot expect all BG-operations in Barnette's and Grünbaum's construction sequence to be basic when G itself contains parallel edges. However, for simple graphs G, the proof of Theorem 17, as given in [5], implicitly shows that basic BG-operations suffice.

Theorem 19 (Barnette and Grünbaum [5])**.** *A simple graph G is 3-connected if and only if G can be constructed from K_4 using* basic *BG-operations.*

For simple graphs on more than 4 vertices, Theorem 19 is in sharp contrast to Tutte's construction sequence, as vertex splittings need parallel edges in intermediate graphs, whereas BG-operations do not (see Figure 3.5).

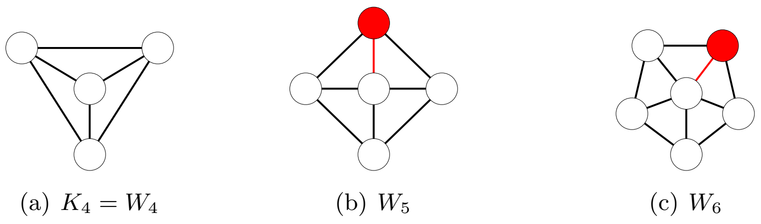

(a) $K_4 = W_4$ (b) W_5 (c) W_6

Figure 3.5: Constructing wheel graphs from K_4 using basic BG-operations. The edges and vertices that are added by BG-operations are depicted in red.

In general, construction sequences using BG-operations are not bound to start with K_4. Titov and Kelmans [38, 70] extended Theorem 19 by proving the existence of a basic construction sequence even when starting with an arbitrary 3-connected graph G_0 instead of K_4, as long as a subdivision of G_0 is contained in G (this is a more general statement, as every 3-connected graph contains a subdivision of K_4; we will prove this in Section 3.3).

Theorem 20 (Kelmans [38], Titov [70])**.** *Let G_0 be a 3-connected graph. A simple graph G is 3-connected and contains a subdivision of G_0 if and only if G can be constructed from G_0 using basic BG-operations.*

3.2.2 Removals

We characterized Tutte's construction sequence by using certain contractions instead of vertex splittings, as those were exactly the inverse operations of vertex splittings (see Theorem 12). Analogously, we want the inverse counterpart of a BG-operation on a 3-connected graph. Let *removing* an edge $e = xy$, $x \neq y$, in a graph be the operation that deletes e followed by smoothing x and y. An edge $e = xy$, $x \neq y$, in a graph G is called *removable* if removing e generates a 3-connected graph. It is easy to see that BG-operations on 3-connected graphs can always be inversed by removing a removable edge.

Lemma 21. *Every BG-operation on a 3-connected graph G can be inversed by removing a removable edge.*

Proof. Let $e = xy$ be the edge that is added by the BG-operation on G and let G' be the graph that is generated by applying it. At most two new vertices, namely x and y, are created by the BG-operation, each of which has degree 3 in G'. Therefore, the removal of e will inverse the BG-operation by smoothing x and y after deleting e. Clearly, e is removable, as G is 3-connected. □

Lemma 21 and Theorem 17 imply the following classic result.

Corollary 22 (Barnette and Grünbaum [5], Tutte [77])**.** Every 3-connected graph on more than 6 edges (not counting self-loops) contains a removable edge.

The converse of Lemma 21 is not true in general, even if every vertex has at least 3 neighbors, as Figure 3.6 shows.

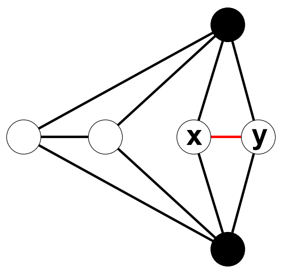
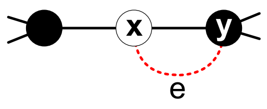


Figure 3.6: The edge xy is removable, since removing xy results in the 3-connected graph K_4. However, the inverse operation of removing xy is not a BG-operation, because it subdivides two parallel edges, inducing the black separation pair.

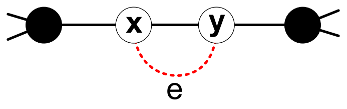
We show under which conditions the inverse operation of removing a removable edge is a BG-operation.

Lemma 23. *The inverse operation O of removing a removable edge $e = xy$ in a graph G is a BG-operation if and only if $|V(G)| = 4$ or $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$ (not counting self-loops). If G is simple, the condition $|V(G)| = 4$ is not needed.*

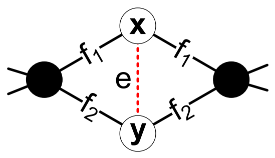
Proof. Let G' be the 3-connected graph that is generated from G by removing e. Assume first that O is a BG-operation and assume further that $|V(G)| > 4$. With Lemma 16, G is 3-connected, which implies $|N(x)| \geq 3$ and $|N(y)| \geq 3$ in G. It



(a) A graph with $|N(x)| = 2$ and a black separation pair. Only the vertex x is deleted when removing e.



(b) A graph with $|N(x)| = |N(y)| = 2$, $|N(x) \cup N(y)| < 5$ and a black separation pair. Both vertices x and y are deleted when removing e.



(c) A graph with $|N(x) \cup N(y)| < 5$ and a black separation pair. Two parallel edges f_1 and f_2 are generated when removing e.

Figure 3.7: Cases in which removing a removable edge e is not a BG-operation.

remains to show that $|N(x) \cup |N(y)| \geq 5$ (we do not count self-loops for neighbors in this proof). 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, $|N(x) \cap N(y)| = 2$. Since $|V(G)| > 4$, a vertex in $N(x) \cap N(y)$ must be adjacent to a vertex that is neither adjacent to x nor to 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 that $|N(x)| < 3$, $|N(y)| < 3$ 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 K_4 is the smallest 3-connected graph, $|V(G)| \neq 4$.

If exactly one vertex, say x, is deleted by the removal of e, let a and b be the two neighbors of x in G that are different from y. Then $y \in \{a, b\}$ must hold, as otherwise O would be a BG-operation and it follows that $|N(x)| < 3$ in G (see Figure 3.7(a)). The case that both vertices x and y are deleted by the removal remains. If x and y are still adjacent after deleting e during its removal, $|N(x)| < 3$, $|N(y)| < 3$ and $|N(x) \cup N(y)| < 5$ holds (see Figure 3.7(b)); this case corresponds to a BG-operation that subdivides identical edges e and f in 15c. Otherwise, let f_1 and f_2 be the distinct edges in which x and y are smoothed, respectively. Then f_1 and f_2 have to be parallel (see Figure 3.7(c)), as otherwise O would be a BG-operation. This implies that $|N(x) \cup N(y)| < 5$. □

Combining Lemmas 21 and 23 with Theorem 17 implies the following theorem.

Theorem 24. *A graph G without self-loops is 3-connected if and only if there is a sequence of removals from G to K_4 on removable edges $e = xy$ with $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$.*

For 3-connected graphs without self-loops, we call the sequence of Theorem 24 a *sequence of removals* or a *top-down* variant of Barnette's and Grünbaum's construction sequence. We remark that the construction sequences of Theorems 17 and 24 can be transformed efficiently into each other (in linear time for suitable representations of the construction sequences) by simply replacing BG-operations with removals and vice versa. If G contains additionally no parallel edges, i. e., if G is simple, combining Lemmas 21 and 23 with Theorem 19 instead of Theorem 17 implies that there is even a sequence of removals from G to K_4 with every intermediate graph being simple, i. e., a basic sequence of removals.

Theorem 24 characterizes the class of 3-connected graphs without self-loops by the existence of a sequence of removals. We show that the same characterization holds if we omit the condition that removed edges have to be removable.

Theorem 25. *A graph G without self-loops is 3-connected if and only if there is a sequence of removals from G to K_4 on edges $e = xy$ with $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$.*

Proof. Clearly, if G is 3-connected, the sequence of removals of the claim exists due to Theorem 24.

Otherwise, let Q be a sequence of removals from G to K_4 on edges $e = xy$ with $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$. Assume to the contrary that G is not 3-connected. Then, as K_4 is 3-connected, there is a removal in Q that is applied on a non-3-connected graph G_{i+1} and generates a 3-connected graph G_i. Let $e = xy$ be the edge in G_{i+1} that is removed by Q. Then e is removable and satisfies $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$ in G_{i+1}. According to Lemma 23, the inverse operation of removing e is a BG-operation. This implies with Lemma 16 that G_{i+1} is 3-connected, which contradicts the choice of G_{i+1}. □

3.2.3 Transformation to Tutte's Sequence

We show that Barnette's and Grünbaum's construction is algorithmically at least as powerful as Tutte's by giving a simple transformation. This transformation can be computed in linear time, assuming a suitable representation of the construction sequences. A suitable representation in linear space $O(m)$ for Tutte's sequence of contractions is just the sequence of the edges that are contracted. Note that the end vertices of an edge may change because of contractions; algorithmically, we can still pinpoint each edge by its unique *label*. We will discuss suitable representations for Barnette's and Grünbaum's construction sequence in Section 3.3.2.

Lemma 26. *Every construction sequence using BG-operations can be transformed in linear time to a sequence of contractions.*

Proof. We transform every BG-operation in reverse order of the given construction sequence to 0, 1 or 2 contractions each. The transformation of each BG-operation can be done in constant time. Let G' be the 3-connected graph on which a BG-operation of the construction sequence is applied and let xy be the edge that is added to G' by this operation. An operation 15a does not induce any contraction. For an operation 15b, we contract exactly one of the two edges that are incident to x but not to y (i. e., either xa or xb in Figure 3.4(b)).

If the operation 15c is applied, let $e = ab$ and $f = vw$ be the edges that are subdivided by x and y, respectively. Both edges share at most one vertex; w. l. o. g. let $a = v$ be that vertex if it exists. We contract the edges xb and yw in arbitrary order (see Figure 3.4(c)).

In all cases, the given contractions inverse the BG-operation except for the edge xy that was added by the BG-operation on G' and which is left over. But additional edges neither harm the 3-connectivity of intermediate graphs nor subsequent contractions. They however generate additional parallel edges and self-loops in the final graph K_4 of the sequence, such that the base graph K_4 has to be replaced by a K_4-multigraph.

By definition of Barnette's and Grünbaum's construction sequence in Theorem 17, the edge that is contracted for an operation 15b is contractible and has end vertices with at least three neighbors each. The last of the two edges that are contracted for an operation 15c has the same properties for the same reason.

Thus, we have found a sequence of contractions from G to a K_4-multigraph, unless the first contraction in the case of an operation 15c generates a graph H that is not 3-connected. We show that this cannot happen. Let H' be the 3-connected graph after the second contraction of the same operation 15c. Then H can be generated from H' by applying operation 15b. According to Lemma 16, H must be 3-connected. $\square$

Lemma 26 will allow us to focus on computing BG-operations only.

3.2.4 Computation in Cubic Time

A straight-forward algorithm to compute a sequence of removals on an input graph G in the case when G is 3-connected is to search and remove iteratively removable edges. But in contrast to the computation of Tutte's construction sequence in Section 3.1.2, this direct approach leads only to a running time of $O(n^3)$ without additional arguments. The reason for the additional factor of n is that vertices of degree 3 in 3-connected graphs are not necessarily incident to a removable edge (see Figure 3.8 for a proof), implying that we cannot bound the number of tested edges until we find a removable one by a constant.

Assume that G is 3-connected. For a simple cubic-time algorithm, we can assume G to contain more than 4 vertices, as otherwise finding a sequence of removals amounts to removing all parallel edges in a K_4-multigraph. We will remove only removable edges. Therefore, the three neighborhood constraints of Theorem 24 are

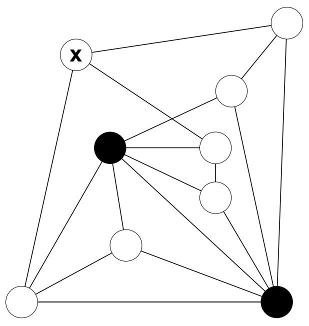


Figure 3.8: A 3-connected graph that contains a vertex x of degree 3 for which no incident edge is removable. Removing each incident edge of x results in the same separation pair, which is depicted in black (although the connected components in which this separation pair splits the graph are different for each edge).

always satisfied until we stop at a K_4-multigraph.

First, the algorithm of Nagamochi and Ibaraki of Theorem 14 is applied to generate a spanning subgraph of G that preserves the graph to be 3-connected or not 3-connected, respectively. If the generated graph is 3-connected, which can be checked efficiently with the algorithm Hopcroft and Tarjan in $O(n)$ time, each edge that is omitted in this step must be removable and can therefore be removed.

For searching the next removable edge, we iterate over all remaining edges and check for each such edge e in $O(n)$ time whether the graph that is generated by removing e is again 3-connected. Removing the first such edge and iterating this process until a K_4-multigraph is generated yields a sequence of removals in time $O(n^3)$.

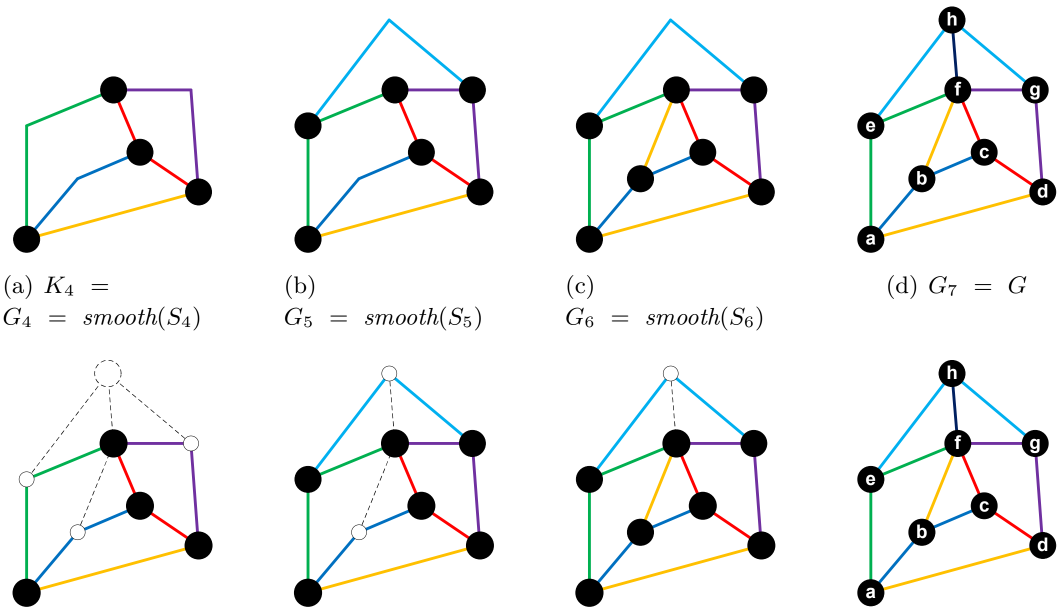
To the best of our knowledge, there is no faster algorithm known for computing a sequence of removals; we will give a quadratic-time algorithm in this chapter and an optimal linear-time algorithm in Chapter 4.

3.3 BG-Paths

Let G be a 3-connected graph without self-loops. According to Theorem 17, there is a construction sequence Q from K_4 to G using BG-operations. Let $G_4, G_5, \ldots, G_z$ with $G_4 = K_4$ and $G_z = G$ be the sequence of 3-connected graphs that is generated by Q and let B_i, $4 \leq i \leq z-1$, be the BG-operation that Q applies on G_i.

Due to Lemmas 21 and 23, we can reverse Q to a sequence of removals by starting with G and removing the added edges of BG-operations in reverse order. Suppose we would delete the added edge of every BG-operation B_i instead of removing it and treat emerging paths containing inner vertices of degree 2 as (topological) edges in G_i (see Figure 3.9). Then iteratively paths are deleted instead of edges being removed and we obtain the sequence of subgraphs $S_z, \ldots, S_4$ of G in which every S_i

is a subdivision of G_i, $S_z = G$ and S_4 is a K_4-subdivision in G. This leads to the following proposition.



(a) $K_4 = G_4 = smooth(S_4)$ (b) $G_5 = smooth(S_5)$ (c) $G_6 = smooth(S_6)$ (d) $G_7 = G$

(e) S_4 (f) S_5 (g) S_6 (h) $S_7 = G$

Figure 3.9: The graphs $G_4, \ldots, G_z$ and $S_4, \ldots, S_z$ of a basic construction sequence of G using BG-operations. On graphs S_i, the dashed edges and vertices are contained in G but not in S_i and vertices depicted in black are *real* vertices. For example, the path $B_4 = e \to h \to g$ is a *BG-path* for S_4, which generates S_5. The *links* in S_5 are the paths B_4, $a \to b \to c$ and the single edges ae, ef, fc, cd, da, fg and gd.

Proposition 27. *Let Q be a construction sequence from a graph H to G using BG-operations. Then G contains a subdivision of H.*

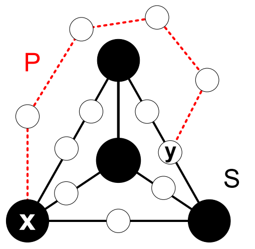
In particular, Proposition 27 implies with Theorem 17 the following result, which was shown by J. Isbell [5].

Lemma 28 (Isbell [5])**.** *Every 3-connected graph contains a subdivision of K_4.*

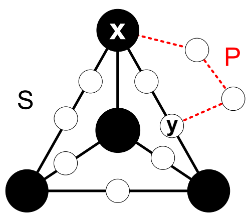
Each S_i is a subdivision of G_i that is contained in G. Conversely, $G_i = smooth(S_i)$ for all $4 \leq i \leq z$, since smoothing a graph is precisely the inverse operation of subdividing a graph in which every vertex has at least 3 neighbors.

From now on, we assume the input graph G to be simple, although all results can easily be extended to non-simple graphs. Note that every S_i must be simple, as $S_i \subseteq G$, while G_i can still contain parallel edges if the construction sequence is not basic (however, G_i does not contain self-loops).

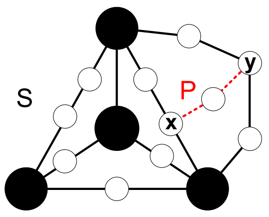
The vertices in S_i that have at least 3 neighbors are special in the sense that they correspond to vertices in G_i. To allow later construction sequences to start with a



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



(b) P is not a BG-path for S, as a link contains x and y with y being no end vertex of that link (see Property 30b).



(c) P is not a BG-path for the K_4-multigraph S, as x and y are inner vertices of two links, respectively, that are parallel (see Property 30c).

Figure 3.10: BG-paths

K_2^3-subdivision instead of a K_4-subdivision, we define these vertices as follows. Let S be a subgraph of G that is a subdivision of either a 3-connected graph or of K_2^3. We say that a vertex v in S is *real* if $N(v) \geq 3$. Let $V_{real}(S)$ be the set of real vertices in S.

Definition 29. Let the *links* of S be the unique paths in S that have real end vertices but contain no other real vertices. Let two links be *parallel* if they share the same end vertices.

The links of S are in one-to-one correspondence to the edges of *smooth*(S). Clearly, the links of a K_2^3-subdivision partition the edge set of the K_2^3-subdivision. If $S \subseteq G$ is a subdivision of a 3-connected graph, the links of S partition $E(S)$ as well, as S is 2-connected, has therefore minimum degree 2 and is not a cycle.

Definition 30. A *BG-path* for S is a path $P = x \rightarrow_G y$ with the following properties (see Figures 3.10(a)-(c)):

(a) $V(P) \cap V(S) = \{x, y\}$

(b) Every link in S that contains both, x and y, contains them as end vertices.

(c) 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.

Recall that a construction sequence of G using BG-operations generates the graphs $G_4, \ldots, G_z$ and that each G_i corresponds to a subgraph S_i of G with S_i being a subdivision of G_i. It is easy to see that $\delta(S_z) \geq 3$ and that every BG-operation on G_i, $4 \leq i \leq z-1$, corresponds to a BG-path for S_i and vice versa. This gives the following result, which is also implicitly proven in [5].

Theorem 31 (implicitly in [5])**.** *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.*

Proof. Assume that G is 3-connected. According to Lemma 1, $\delta(G) \geq 3$ holds. Due to Theorem 17, there is a construction sequence using BG-operations from K_4 to G. Thus, there is also a construction sequence using BG-paths from a K_4-subdivision that is in G to a G-subdivision that is in G. The latter must be G itself.

Assume that $\delta(G) \geq 3$ holds and that G can be constructed from a K_4-subdivision in G by adding BG-paths. Thus, there is a construction sequence using BG-operations from K_4 to $smooth(G)$. Due to Lemma 16, $smooth(G)$ is 3-connected. Since $\delta(G) \geq 3$, $smooth(G) = G$ and it follows that G is 3-connected. □

We can therefore try to compute Barnette's and Grünbaum's construction sequence of a 3-connected graph by taking a suitable K_4-subdivision in G and adding iteratively BG-paths. Note that adding a BG-path P to a subgraph S_i implies that the end vertices of P are real in S_{i+1}. Compared to BG-operations, the BG-path construction has the advantage that every S_i is a subgraph of G. This allows to see a construction sequence of G as decomposition of G.

We call the sequence of Theorem 31 for 3-connected graphs without self-loops a *construction sequence using BG-paths*. To establish consistency with BG-operations, we define the addition of a BG-path to be *basic* if it does not create a new parallel link.

3.3.1 Prescribing a Base Graph

Theorem 31 does not specify on which K_4-subdivision in G the construction sequence starts. The more general Theorem 20, rephrased to BG-paths, does also not state on which G_0-subdivision in G we have to start for a given 3-connected graph G_0. We want to find these base graphs in order to compute the corresponding construction sequences bottom-up.

However, the proof of Theorem 20 in [38] and the implicit proof of Theorem 31 in [5] show only for a very special subdivision H in G that a construction sequence from H to G using BG-paths exists. In fact, both proofs choose H as a subdivision that contains the maximum number of edges in G. The construction sequence is then generated by iteratively adding longest BG-paths. Unfortunately, computing these depends heavily on solving the longest paths problem, which is known to be NP-hard even for 3-connected graphs [24].

Suppose we choose some subdivision H (of K_4 or, more general, of a 3-connected graph G_0) in advance; we say that H is *prescribed*. Is it possible to strengthen Theorems 20 and 31 to a construction sequence using BG-paths that starts with H? Such a result would provide an efficient computational approach to construction sequences, since it would allow to search the neighborhood of H in G for BG-paths, yielding a new prescribed subdivision of a 3-connected graph.

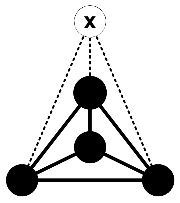


Figure 3.11: Every possible BG-path adds a parallel link to the black subgraph.

However, when restricted to basic operations, it is not possible to prescribe H, as the following minimal counterexample shows: Consider the graph G in Figure 3.11, which consists of a K_4-subdivision H (here just a K_4) that is depicted in black and an additional vertex x that is adjacent to three vertices of H. Then every BG-path for H will create a parallel link, namely a path of length two with x as inner vertex.

What happens if we drop the condition that construction sequences have to be basic? The following theorem shows that at this expense we can indeed start a construction sequence using BG-paths from any prescribed subdivision.

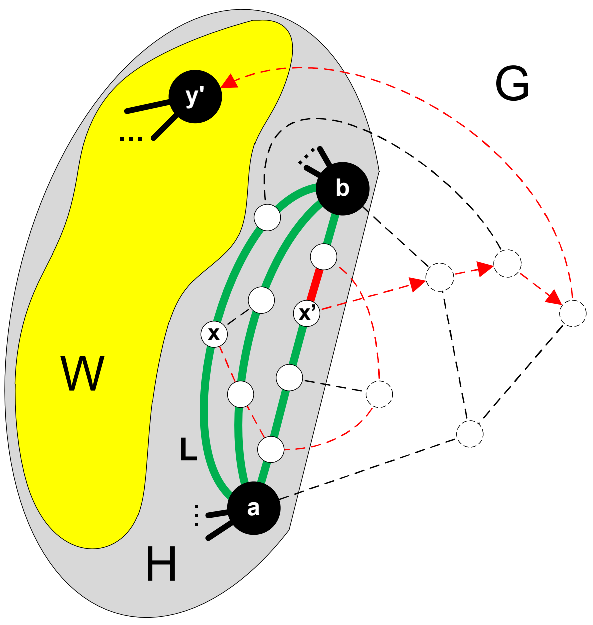


Figure 3.12: The case $H \neq smooth(H)$ with H being a subdivision of a 3-connected graph. Dashed edges are contained in $E(G) \setminus E(H)$, while arrows depict a BG-path $x' \rightarrow_{G \setminus V(H \setminus \{x',y'\})} y'$.

Theorem 32. *Let G be a simple 3-connected graph and $H \subset G$ such that H is a subdivision of either K_2^3 or of a 3-connected graph. Then there is a BG-path for H in G. If H is a K_2^3-subdivision, there is a BG-path for H in G that generates a K_4-subdivision. Moreover, for every link L in H of length at least 2, L or a parallel link of L contains an inner vertex on which a BG-path for H starts.*

Proof. We distinguish two cases.

- $H \neq smooth(H)$.
  Then H contains a link of length at least 2 and we pick an arbitrary such link, say $L = a \rightarrow_H b$. Let x be an inner vertex of L and let I be the union of inner vertices of all parallel links of L. We show that there is a vertex in I on which a BG-path for H starts. By the 3-connectivity of G, the graph $G \setminus \{a, b\}$ is connected.

  Assume that H is a subdivision of a 3-connected graph. Since H contains at least four real vertices, there exists a path $P = x \rightarrow_{G\setminus\{a,b\}} y$ with $y \in V(H) \setminus I$ (see Figure 3.12). The path P has Property 30b. Let x' be the last vertex in P that is contained in I and let y' be the first vertex in P that is contained in $V(H) \setminus I$. Then the path $x' \rightarrow_P y'$ has Properties 30a–c and is a BG-path for H.

  Assume otherwise that H is a K_2^3-subdivision. Then at least one other parallel link of L in H has length at least 2, as G is simple. Therefore, a path $P = x \rightarrow_{G\setminus\{a,b\}} y$ with $y \in I \setminus V(L)$ exists. Let x' be the last vertex in P that is contained in L and let y' be the first vertex in P that is contained in $I \setminus V(L)$. Then the path $x' \rightarrow_P y'$ has Properties 30a–c, as $|V_{real}(H)| < 4$, and is a BG-path for H that generates a 3-connected K_4-subdivision.

- $H = smooth(H)$.
  Then H consists only of real vertices and contains no link of length at least 2. Because G is simple, H is simple and cannot be a K_2^3-subdivision. Since $H \subset G$, there must be a vertex in $V(G)\setminus V(H)$ or an edge in $E(G)\setminus E(H)$. First, assume that there is a vertex $x \in V(G) \setminus V(H)$. Then, by the 2-connectivity of G and Fan Lemma 3, we can find a path $P = y_1 \rightarrow_G y_2$ with $P \cap H = \{y_1, y_2\}$ that contains x as an inner vertex. The path P has Properties 30a–c, because every link in H is an edge. Now suppose that $V(G) = V(H)$ and let e be an edge in $E(G)\setminus E(H)$. Then e must be a BG-path for H, since both end vertices are real. □

Therefore, we can start a construction sequence using BG-paths from every K_4-subdivision in G and, more generally, from every subgraph in G that is a subdivision of either a 3-connected graph or of K_2^3. As BG-operations preserve 3-connectivity due to Lemma 16 (and, thus, BG-paths preserve subgraphs to be subdivisions of 3-connected graphs), Theorem 32 implies the following result.

Theorem 33. *A simple graph G is 3-connected if and only if $\delta(G) \geq 3$ and G can be constructed from* each *K_4-subdivision in G by adding BG-paths.*

Note that Theorem 33 is stronger than Theorem 31 for simple 3-connected graphs G, as the construction can start at any K_4-subdivision in G.

We showed that construction sequences that start at prescribed base graphs are in general not basic. However, at the expense of augmenting our set of allowed operations, we can establish all operations to be basic. In Theorem 32, non-basic BG-paths occur only in the case $H = \mathit{smooth}(H)$, as in the other case a BG-path is found that starts at an inner vertex of a link. Let $H = \mathit{smooth}(H)$. If additionally $V(H) = V(G)$, every BG-path must be an edge and is basic, as G is simple. We conclude that non-basic BG-paths are only needed for the subcase $V(H) \subset V(G)$ of $H = \mathit{smooth}(H)$ in Theorem 32, in which we picked a BG-path that contains an inner vertex x in $V(G) \setminus V(H)$.

In this case, we know by the Fan Lemma 3 and the 3-connectivity of G that there are three internally vertex-disjoint paths from x to real vertices in H such that only the three end vertices are in H. Adding these paths to H is called an *expand* operation. Clearly, the expand operation is basic, because each new link ends on the new real vertex. We state the corresponding basic operation for BG-operations.

(15d) create a new vertex that is adjacent to exactly 3 distinct old vertices

Note that the Operation 15d can be interpreted as concatenation of the BG-operations 15a and 15b. We obtain the following theorem to avoid non-basic operations.

Theorem 34. *Let G be a simple graph and let H be a subdivision of a 3-connected graph. Then*

$$\begin{aligned} & G \text{ is 3-connected and } H \subseteq G \\ \Leftrightarrow\ & \delta(G) \geq 3 \text{ and there is a construction sequence from} && (*) \\ & H \text{ to } G \text{ using BG-paths} \\ \Leftrightarrow\ & \delta(G) \geq 3 \text{ and there is a basic construction sequence from} && (**) \\ & H \text{ to } G \text{ using BG-paths and the expand operation} \end{aligned}$$

Proof. Let G be 3-connected and $H \subseteq G$. Then $\delta(G) \geq 3$ holds with Lemma 1 and if $H = G$, the desired construction sequences (*) and (**) are empty and exist. If $H \subset G$, Theorem 32 can be iteratively applied to generate a sequence (*). We do the same to generate a sequence (**) with the exception that for every subcase $V(H) \subset V(G)$ of $H = \mathit{smooth}(H)$ of Theorem 32 an expand operation is inserted.

For the sufficiency part, both construction sequences (*) and (**) imply $H \subseteq G$, since only paths are added to construct G. With Lemma 5, Operation 15d preserves 3-connectivity. Thus, the expand operation preserves generated subgraphs to be subdivisions of 3-connected graphs, as do BG-path additions. Thus, G is for both sequences (*) and (**) a subdivision of a 3-connected graph and therefore 3-connected because of $\delta(G) \geq 3$. □

For the linear-time algorithm in Chapter 4, we need the fact that it is possible to start with a K_2^3-subdivision instead of a K_4-subdivision.

Theorem 35. *A simple graph G is 3-connected if and only if $\delta(G) \geq 3$ and G can be constructed from a K_2^3-subdivision in G by adding BG-paths such that the first BG-path generates a K_4-subdivision.*

Proof. Let G be 3-connected. According to Lemma 1, $\delta(G) \geq 3$ holds. With Lemma 28, G contains a K_4-subdivision and therefore also a K_2^3-subdivision S_3. Adding iteratively BG-paths to S_3 with Theorem 32 gives the desired sequence.

Let $\delta(G) \geq 3$ and let S_3 be the K_2^3-subdivision on which the given construction sequence Q of the claim starts. Since G is simple and $\delta(G) \geq 3$, $S_3 \neq G$ and Q is not empty. Let S_4 be the K_4-subdivision that is generated by the first BG-path in Q by assumption. According to Lemma 31, the remaining BG-paths in Q prove that G is 3-connected. □

Note that we can replace "a K_2^3-subdivision" in Theorem 35 by "each K_2^3-subdivision", as done for Theorem 33.

3.3.2 Representations

We discuss representations of the previously mentioned construction sequences. In the following, graphs are represented with adjacency lists. We will often identify the vertices and edges of a graph by their *labels*; these labels have to be unique in the graph. Unless a vertex or an edge is explicitly relabeled, its label does not change; this holds in particular for the label of an edge whose end vertices are replaced due to, say, contractions.

Recall that a sequence of contractions can be represented by just giving the labels of the contracted edges in the right order. However, the sequence of removals does not have a similar representation, as removing an edge may create new edges for which we have to specify labels.

We discuss representations of Barnette's and Grünbaum's construction sequence in the bottom-up variant. Lemmas 21 and 23 ensure analogue representations for the top-down variant (i. e., the sequence of removals). We can either use BG-operations or BG-paths. Let Q be a construction sequence using BG-operations. We can represent Q by storing the base graph $G_4 = K_4$ and the sequence $B_4, \ldots, B_{z-1}$ of BG-operations that Q applies. Unfortunately, the graphs $G_4, \ldots, G_{z-1}$ are not necessarily subgraphs of G, so we have to take care of relabeling vertices and edges in every BG-operation.

For the BG-operation B_i on G_i, we specify the label of every edge e that is subdivided by B_i, followed by the new labels that are assigned to the new vertex of degree 2 and to one of the two new edges in G_{i+1}; the other new edge in G_{i+1} keeps the label of the deleted edge e. Note that this is only needed if B_i is either an operation 15b or 15c.

If B_i is an operation 15a or 15b, we specify the labels of all end vertices of its added edge in G_{i+1} that have been already contained in G_i. Additionally and in every case, we assign a new label to the added edge in G_{i+1}. Finally, we have to impose the constraint that all labels are chosen such that G_z is not only isomorphic but also identical to G, meaning that corresponding vertices and (for convenience) edges of G_z and G have the same label. Otherwise, we would have to solve the graph isomorphism problem to check that Q really constructs G.

While this representation allows to check the validity of the sequence (i. e., every operation on being a BG-operation) very easily, the relabeling issues make it cumbersome. On the other hand, BG-paths allow to represent Q without relabeling issues, as every intermediate graph S_i and every BG-path is a subgraph of G. We just store $S_4 \subset G$ and the iteratively added BG-paths $P_4, \ldots, P_{z-1}$. Hence, we can represent Barnette's and Grünbaum's construction sequence of a graph G in the following two ways.

- *Edge representation*: Store G_4 and the sequence $B_4, \ldots, B_{z-1}$ of BG-operations and specify new and old labels for every B_i such that G_z and G are labeled the same.
- *Path representation*: Store S_4 and the sequence $P_4, \ldots, P_{z-1}$ of BG-paths as subgraphs of G.

Both representations refer to the same sequence of graphs $G_i = \mathit{smooth}(S_i)$ and need space linearly dependent on the graph size, i. e., $\Theta(m)$ space in the uniform cost model, as G is connected. We will show in Section 3.6.2 how the validity of path representations can be checked in a simple way. The next lemma states that it does not matter which of the two representations we use.

Lemma 36. *The edge and path representations of Barnette's and Grünbaum's construction sequence can be transformed into each other in $O(m)$ time.*

Proof. Let Q be an edge representation of the construction sequence. If an operation B_i subdivides an edge e, we define $\beta(e, B_i)$ to be the edge that gets a new label. Let e be the added edge of an operation B_i in Q. Exploiting the duality of BG-operations and BG-paths, the edge e corresponds to the BG-path P_i, which will be subdivided by inserting exactly $|P_i| - 1$ real vertices in the path representation. To compute the

BG-path P_i from e, we have to keep track of the at most $|P_i| - 1$ operations that subdivide e and glue the subdivided parts back together.

Whenever an operation B_j in Q subdivides e, we store a pointer at $\beta(e, B_j)$ to e. Moreover, for every edge f that points to e and is subdivided by an operation B_k, we store a pointer at $\beta(f, B_k)$ to e. In both cases, we append $\beta(e, B_j)$ (respective $\beta(f, B_k)$) to a list stored on the edge e. Therefore, we keep track of all edges $\beta(e, B_j)$ and $\beta(f, B_k)$ that correspond to BG-paths that subdivide P_i by inserting a new real vertex. Eventually, we get all the edges in which P_i got subdivided by augmenting the list that is stored on e with e itself. Hence, we have computed the set of edges that P_i consists of. Since G_z has the same labeling as G, the computed set of edges and $E(P_i)$ have the same labels.

The list of edges is not necessarily in the order of appearance in P_i, but this can be easily fixed in time $O(|P_i|)$ by temporarily storing the incidence information of every vertex in P_i and extracting the BG-path P_i from a vertex with degree one. In order to compute S_4, we analogously maintain pointers for each edge of G_4 and get the links of S_4. Since the links of S_4 together with $P_4, \dots, P_{z-1}$ partition $E(G)$, the running time is $O(m)$.

Let the path representation, i. e., S_4 and the sequence $P_4, \dots, P_{z-1}$ of BG-paths be given. We remove BG-paths in reversed order from G. Note that the first removal is well-defined, as P_{z-1} is an edge. If an end vertex x in the removal of a BG-path is deleted due to smoothing, we also smooth x in the BG-path (or the link of S_4) that contains x as inner vertex. This way, every P_i will be an edge immediately before removing it. Note that we can find the BG-path (and, analogously, the link of S_4) that contains x as an inner vertex in $O(1)$ time by storing a pointer from each inner vertex of P_i to P_i in advance.

This procedure passes through the graph sequence $G_z, \dots, G_4$. We use the labels of G for G_z. If both end vertices x and y of P_i are still real after the removal of P_i, we can keep their labels and construct the inverse BG-operation 15a. Otherwise, at least one end vertex of P_i, say x, was deleted during the removal, as it was smoothed into an edge e. For the inverse BG-operation 15b or 15c, it remains to assign a label to e. Let f_1 and f_2 be the incident edges of x before the removal. We assign the label of f_1 to e. Thus, each of the BG-operations 15a–c can be constructed in constant time. □

3.4 Transformations

We summarize the given characterizations of simple 3-connected graphs by construction sequences and group similar ones. Whenever not otherwise possible, we will refer to a construction sequence of a certain type A as (the) *sequence* A, although there might be more than one sequence of that type.

Theorem 37. *The following statements are equivalent.*

A simple graph G is 3-connected (3.0)

$\Leftrightarrow$ *There is a sequence of vertex splittings from a K_4-multigraph to G (see Theorem 9)* (3.1)

$\Leftrightarrow$ *There is a sequence of contractions from G to a K_4-multigraph on* contractible *edges $e = xy$ with $|N(x)| \geq 3$ and $|N(y)| \geq 3$ (see Theorem 12)* (3.2)

$\Leftrightarrow$ *There is a sequence of contractions from G to a K_4-multigraph on edges $e = xy$ with $|N(x)| \geq 3$ and $|N(y)| \geq 3$ (see Theorem 13)* (3.3)

$\Leftrightarrow$ *There is a sequence of BG-operations from K_4 to G (see Theorem 17)* (3.4)

$\Leftrightarrow$ *There is a sequence of* basic *BG-operations from K_4 to G (see Theorem 19)* (3.5)

$\Leftrightarrow$ *There is a sequence of removals from G to K_4 on* removable *edges $e = xy$ with $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$ (see Theorem 24)* (3.6)

$\Leftrightarrow$ *There is a sequence of removals from G to K_4 on edges $e = xy$ with $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$ (see Theorem 25)* (3.7)

$\Leftrightarrow$ *$\delta(G) \geq 3$ 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 (see Theorem 35)* (3.8)

$\Leftrightarrow$ *$\delta(G) \geq 3$ and there is a basic construction sequence from a (resp. each) K_4-subdivision in G to G using BG-paths and the expand operation (see Theorem 34)* (3.9)

$\Leftrightarrow$ *$\delta(G) \geq 3$ and there is a sequence of BG-paths from a K_4-subdivision in G to G (see Theorem 31)* (3.10)

$\Leftrightarrow$ *$\delta(G) \geq 3$ and there is a sequence of BG-paths from* each *K_4-subdivision in G to G (see Theorem 33)* (3.11)

We give several efficient transformations between these construction sequences.

Lemma 38. *The sequences* (3.9) *and* (3.10) *can be transformed into each other in $O(m)$ time.*

Proof. If a sequence (3.9) is given, the three internally vertex-disjoint paths of each expand operation can be easily split into two subsequent BG-paths, the first of which is possibly a non-basic BG-operation.

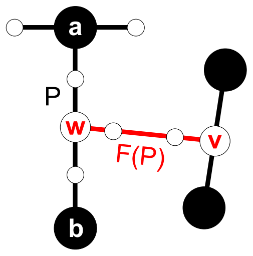


Figure 3.13: No expand operation can be formed.

Let a sequence (3.10) be given. With Lemma 36, we can assume that this sequence is given in the path representation Q. We will rearrange the order of BG-paths in Q to generate pairs of subsequent BG-paths that can be concatenated to expand operations. For each BG-path (or link of S_4) P, its position in Q and a pointer to the first BG-path $F(P)$ in Q that ends at an inner vertex of P (if that path exists) is stored. We define the position of each link in S_4 as 0. Note that $F(P)$ for all P can be precomputed in $O(m)$ by simulating Q once.

Performing a bucket sort on the lower end vertices of each BG-path and link of S_4 (lower in any given total order on $V(G)$) followed by a stable bucket sort on the remaining end vertices gives a list of paths sorted in lexicographic order of their end vertices. This list can be used to efficiently group paths that have the same end vertices.

Let R_{ab} be the set of all BG-paths and links in S_4 having end vertices a and b. We apply the following rule: If a path $P \in R_{ab}$ has length one and does not have the first position of all paths in R_{ab}, we append it to the end of the construction sequence and delete it from R_{ab}. This does not harm the construction sequence, since a and b must already be real vertices and P has no inner vertices.

Now the path with the first position in R_{ab} cannot lead to a non-basic operation and causes a and b to be real. We consider every other path $P \in R_{ab}$, which is possibly non-basic, but must contain an inner vertex w that is an end vertex of the BG-path $F(P)$. Without harming the construction sequence, P can be moved to the position of $F(P)$, since a and b are real and no inner vertex of P is part of a BG-path before $F(P)$ is added.

Let v be the end vertex of $F(P)$ different from w. If v is real immediately before $F(P)$ is added, we can glue P and $F(P)$ together to an expand operation, which is basic due to its new vertex w. Otherwise, v is an inner vertex of a link (see Figure 3.13) and P and $F(P)$ can be replaced with either the BG-paths $v \rightarrow_{P \cup F(P)} a$ and $b \rightarrow_{P \cup F(P)} w$ or the BG-paths $v \rightarrow_{P \cup F(P)} b$ and $a \rightarrow_{P \cup F(P)} w$. Both BG-paths are basic, since they contain end vertices of degree 2. □

Lemma 39. *Let Q be one of the sequences (3.4)–(3.7) and (3.10). There are algorithms that transform Q to each of the sequences (3.1)–(3.4) and (3.6)–(3.10) in time*

$O(m)$. The transformations to sequence (3.4), (3.6), (3.7) and (3.10), respectively, preserve the number of operations.

Proof. According to Lemmas 7 and 11, the sequences (3.1), (3.2) and (3.3) can be transformed to each other in time $O(m)$ (note that (3.2) and (3.3) represent the same sequence). Analogously, the sequences (3.4), (3.6) and (3.7) can be transformed to each other in time $O(m)$ with Lemmas 21 and 23. A sequence (3.5) is a special sequence (3.4) and can therefore be transformed to the sequences (3.6) and (3.7) as well. With Lemma 36, we can transform the sequences (3.4) and (3.10) to each other in time $O(m)$. All these transformations preserve the number of operations by construction.

It remains to show how sequences (3.2), (3.8) and (3.9) can be computed. According to Lemma 26, sequence (3.4) can be transformed to a sequence (3.2) in time $O(m)$. With Lemma 38, sequence (3.10) can be transformed to a sequence (3.9) in time $O(m)$. To obtain a sequence (3.8) from sequence (3.10), we just decompose the K_4-subdivision of sequence (3.10) in time $O(m)$ 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, whose addition generates a K_4-subdivision. □

For simple graphs G, there is always a basic construction sequence (3.5) using BG-operations (see Theorem 19). However, we want to start from a prescribed K_4-subdivision, for which non-basic operations are in general unavoidable due to the counterexample in Figure 3.11. We show that this is not a limitation: For simple graphs, a non-basic construction sequence can be transformed to a basic one in linear time.

Lemma 40. *Every sequence (3.10) can be transformed to a sequence (3.5) in $O(m)$ time such that the number of operations is preserved.*

Proof. Let Q be the given sequence (3.10) in the path representation. We will rearrange the order of BG-paths in Q and modify S_4 in linear time to avoid parallel links. This prevents the number of BG-paths. The algorithm of Lemma 36 can then be used for an efficient transformation to a sequence of BG-operations in which every intermediate graph is simple.

As for Lemma 38, we use the following notation. For each BG-path (or link of S_4) P, its position in Q and a pointer to the first BG-path $F(P)$ in Q that ends at an inner vertex of P (if that path exists) is stored. We define the position of each link in S_4 as 0. Note that $F(P)$ for all P can be precomputed in $O(m)$ by simulating Q once.

Let X be the union of the links of S_4 and the BG-paths in Q. We define the relation $\prec$ on X such that, for the paths $P_1 \in X$ and $P_2 \in X$, $P_1 \prec P_2$ if the position of P_1 strictly precedes the position of P_2. Let W be any linear extension of $\prec$.

Performing a bucket sort on the lower end vertices of each path in W (lower in any given total order on $V(G)$) followed by a bucket sort on the remaining end vertices gives a list of paths R sorted in lexicographic order of their end vertices. This list can be used to efficiently group paths that have the same end vertices. Let $R_{a,b} = P_1, \dots, P_k$ be the list of paths in R with end vertices a and b. Since bucket sort is stable, $P_1 \prec \dots \prec P_k$ holds.

We want to avoid parallel links between each two vertices a and b. It is sufficient to consider a list $R_{a,b}$ only if it contains more than one chain, as otherwise no parallel link $a \rightarrow_G b$ can occur in the sequence. For every list $R_{a,b} = P_1, \dots, P_k$, we compute the path $P_{\min} \in R_{a,b}$, for which $F(P_{\min})$ exists and is minimal with respect to $\prec$. This can be done in time $O(k)$ by passing once through the list $R_{a,b}$. If $P_{\min} \neq P_1$, we swap $P_{\min}$ and P_1 in W. If additionally P_1 is a link in S_4, we replace P_1 in S_4 with $P_{\min}$. Both does not harm the construction sequence, as $P_1 \prec P_{\min} \prec F(P_{\min})$ and all paths in W remain BG-paths (except for the possible new path in S_4).

For each remaining path $P \in R_{a,b} \setminus \{P_{\min}\}$, for which $F(P)$ exists, we move P to the position immediately before $F(P)$. This does not harm BG-paths in the construction sequence, as $P_{\min}$ in W implies that a and b are real and every BG-path containing an inner vertex of P does not precede $F(P)$ in W.

If there is a path in $R_{a,b}$, for which no $F(P)$ exists, it must be a unique path of length one, as Q does not subdivide it and G is simple. Such paths can be safely moved to the end of W, as $R_{a,b}$ contains more than one path. We conclude that W is still a construction sequence when starting with the modified subgraph S_4 and that W avoids any parallel link, as each link L of length at least two is subdivided by a BG-path before a parallel link of L is added. □

We combine the previous lemmas in the following theorem.

Theorem 41. *Every sequence of (3.4)–(3.10) can be transformed to each of the sequences (3.1)–(3.10) in $O(m)$ time.*

Proof. According to Lemmas 39 and 40, it suffices to show that every sequence (3.8) and (3.9) can be transformed to a sequence (3.10) in time $O(m)$. As G cannot be a K_2^3-subdivision, every sequence (3.8) adds at least one BG-path. By definition, the first added BG-path generates a K_4-subdivision S_4; the remaining part of sequence (3.8) applied on S_4 is a sequence (3.10). Note that the transformed sequence will have one operation less. Due to Lemma 38, every sequence (3.9) can be transformed to a sequence (3.10) in time $O(m)$. □

Theorem 41 allows us to focus only on the computation of a special sequence, e. g., on a sequence (3.8) or a sequence (3.10).

3.4.1 Size of Sequences

Let the *size* of a construction sequence be the number of operations it applies. Every sequence of (3.1)–(3.3) has size $n - 4$, as it must contain exactly $n - 4$ contractions to a K_4-multigraph (respectively, vertex splittings from a K_4-multigraph). This implies that the K_4-multigraph in every such sequence contains exactly $m-n+4$ edges, as every contraction reduces the number of edges by one. We show a complementary result for the size of Barnette's and Grünbaum's construction sequence.

Lemma 42. *Every sequence of* (3.4)–(3.7) *and* (3.10) *contains exactly* $m-n-2$ *operations. Every sequence* (3.8) *contains exactly* $m-n-1$ *operations.*

Proof. It suffices to show the first claim for each sequence (3.4), as Lemmas 39 and 40 preserve the number of operations, respectively. The second claim follows directly from the fact that the transformation of a sequence (3.10) to a sequence (3.8) adds exactly one BG-path. Let a, b and c denote the number of BG-operations in a sequence (3.4) 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 $a+b+c=m-n-2$. □

3.5 A General Approach in Quadratic Time

We show how to compute a sequence (3.10) in time $O(n^2)$. This extends to an algorithm that tests a graph on being 3-connected in the same time. We will make this algorithm certifying in the next section; as certificate we will use the construction sequence. The running time is dominated by the time needed for finding the construction sequence and every improvement made there will automatically result in a faster 3-connectivity test. Assume that G' is a simple connected input graph with n vertices (G' does not necessarily have to be 2-connected). We follow the steps:

- Apply the linear-time algorithm of Nagamochi and Ibaraki (see Theorem 14) to G' in order to generate a graph G with $O(n)$ edges ($n=|V(G')|$).
- Try to compute a K_4-subdivision in G in $O(n)$ time.
  - Success: Let S_4 be the K_4-subdivision.
  - Failure: Return a separation pair or cut vertex.
- Try to compute a sequence from the prescribed subgraph S_4 to G using BG-paths in $O(n^2)$ time.
  - Success: Return the construction sequence.
  - Failure: Return a separation pair or cut vertex.

According to Theorem 14, the graph G has only $O(n)$ edges and is 3-connected if and only if the input graph G' is 3-connected. We first describe how to find a K_4-subdivision in G by one depth-first search (DFS) [43, 64, 67]. As a byproduct, the DFS can sort out graphs G' with $\delta(G')<3$ (as this implies $\delta(G)<3$) and return a cut vertex or separation pair in that case. Note that the algorithm generalizes

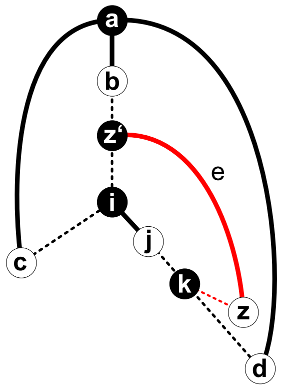


Figure 3.14: Finding a K_4-subdivision. Dashed edges depict (possibly empty) paths, arcs depict backedges.

easily to non-simple and non-connected input graphs, as the DFS can also be used to eliminate self-loops and parallel edges and to test the graph on being connected.

Definition 43. Let T be a DFS-forest of a simple graph G. An edge in $E(G) \setminus E(T)$ is called a *backedge*. For a backedge e, let $s(e)$ and $t(e)$ be the two end vertices of e such that $s(e)$ is a proper ancestor of $t(e)$ in T. Let a backedge e *enter* a subtree T' of a tree if $s(e) \notin V(T')$ but $t(e) \in V(T')$.

In cases where no orientation is given but needed, we assume e to be oriented from $s(e)$ to $t(e)$. Note that this orientation differs from standard graph theory notation.

Lemma 44. *Let G be a simple connected graph on at least 4 vertices. There is a DFS-based algorithm that computes either a K_4-subdivision, a cut vertex or a separation pair in G in time $O(n+m)$.*

Proof. Let T be a DFS-tree of G and let a (respectively, b) be the vertex in T that is visited first (respectively, second). If a has more than one child, $\{a\}$ must be a cut vertex, as T is a DFS-tree. If a has exactly one child and b has more than one child, $\{a, b\}$ must be a separation pair for the same reason. In both cases, we output the cut vertex respective the separation pair.

Otherwise, both, a and b, have exactly one child. We choose two arbitrary neighbors c and d of a that are different from b (see Figure 3.14). W.l.o.g., let d be visited later by the DFS than c. Let i be the least common ancestor of c and d in T. Then $i \neq b$, as b has exactly one child in T. Since G is simple, $d \neq i$ holds. Let j be the child of i that is contained in the path $d \rightarrow_T i$.

If G is 3-connected, there must be a backedge e that starts on an inner vertex z' of $a \rightarrow_T i$ and enters $T(j)$, as otherwise a and i would form a separation pair that separates b and d. Clearly, such an edge e can be found in time $O(n+m)$; if it does not exist, we output the separation pair $\{a, i\}$. Otherwise, let z be the end vertex of

e in $T(j)$ and let k be the first vertex of the path $z \to_T j$ that is contained in $d \to_T j$. Each of the three backedges ac, ad and e close a cycle when added to T, resulting in six internally vertex-disjoint paths connecting the vertices in $\{a, z'\}$, $\{z', i\}$, $\{i, k\}$, $\{k, a\}$, $\{z', k\}$ and $\{a, i\}$, respectively. Thus, we have found a K_4-subdivision in G with real vertices a, z', i and k. □

Assume that we have found a K_4-subdivision H in G. We now show how to carry out the last step of the algorithm. In order to find the construction sequence, we use the path representation and try to find iteratively BG-paths along the lines of Theorem 32.

Lemma 45. *Let H be a subdivision of a 3-connected graph that is contained in a simple 3-connected graph G. There is a algorithm that computes a BG-path for H in time $O(n + m)$.*

Proof. We compute the links of H in $O(n+m)$. For every link L, we store a pointer to L on each inner vertex of L in $O(n + m)$ total time. Moreover, we maintain a pointer on each link that points to its end vertices. It remains to show how to find a BG-path along the lines of Theorem 32. We can easily test in $O(n + m)$ whether $H \neq smooth(H)$ by checking whether there is a vertex x of degree 2 in H.

In case $H \neq smooth(H)$, we pinpoint the link $L = a \to_H b$ of H that contains x in constant time. We compute the path $P = x \to y'$ by temporarily deleting a and b and performing a DFS on x that stops on the first vertex $y' \in V(H)$ that is not contained in a parallel link of L (including L). We can check whether y' is contained in a parallel link of L in constant time by comparing the end vertices of its containing link (if exists) with a and b. Thus, the path $x' \to_P y'$ with x' being the last vertex contained in a parallel link of L is a BG-path and can be found in $O(n + m)$.

In case $H = smooth(H)$, we delete temporarily all edges in $E(H)$ and start a DFS on a vertex $x \in V(H)$ that has an incident edge in the remaining graph. The traversal is stopped on the first vertex $y \in V(H) \setminus x$. Let T be the created DFS-tree. Then the path $x \to_T y$ is the desired BG-path. □

By iterating the algorithm of Lemma 45 on G, sequence (3.10) can be found in time $O(n^2)$, as G contains only $O(n)$ edges. This assumes G to be 3-connected. If G is not 3-connected, a sequence (3.10) cannot exist due to Theorem 37.

In that case, it remains to show that we can always find a separation pair or a cut vertex. For some subgraph $H \subset G$, the DFS starting at vertex x in Lemma 45 fails to find a new BG-path for H. If $H \neq smooth(H)$, the end vertices of the link that contains x form a separation pair. Otherwise, $H = smooth(H)$ and the vertex on which we started a DFS must be a cut vertex. Thus, if G is not 3-connected, the algorithm returns either a separation pair or a cut vertex. Using Theorem 41, we obtain the following theorem.

Theorem 46. *There is an algorithm that tests the* 3*-connectivity of a simple graph* G *in time* $O(n^2)$*, returns each of the sequences* (3.1)–(3.10) *if* G *is* 3*-connected and returns a cut vertex or a separation pair otherwise.*

3.6 Certifying Algorithms

A *certifying* algorithm is an algorithm that produces, with each output, a certificate that the particular output has not been compromised by a bug [44]. Certifying algorithms therefore give a certificate of correctness for every instance along with their output. A user or program that attempts to check the correctness of such an output is called a *checker*. A checker gets the unmodified input data and the output and certificate of the certifying algorithm. We list key properties of certificates and checkers that are mentioned in [44].

- A checker must detect possible false certificates, as not only the output but also the certificate may have been compromised by a bug.
- The checker has to be as simple as possible, since a complicated checker could itself have bugs. Conversely, the certifying algorithm and its proof of correctness can be complicated and even error-prone.
- To have confidence in the output of a certifying algorithm, it is important for the user to understand why the certificate proves what it claims to.
- The certificate must always exist, but the proof of this is of no concern to the checker.

Achieving certifying algorithms is a major goal for problems where the fastest solutions known are complicated and difficult to implement. Testing graphs on 3-connectivity is such a problem, but no certifying linear-time algorithms are known. As a first step towards a linear-time certifying algorithm, we want to make the algorithm of Theorem 46 certifying.

If the input graph is 3-connected, we use the construction sequence (3.10) as a certificate; we will give a detailed description of its verification in Section 3.6.2. If the input graph is not 3-connected, the algorithm of Theorem 46 returns either a cut vertex or a separation pair. Cut vertices and separation pairs prove that the input graph is not 3-connected. Although they are not difficult to check, using them would not satisfy the need for checkers to be as easy as possible. We will instead use slightly simpler certificates for vertex cuts and edge cuts in the next section.

Note that the first step of the algorithm, i. e., the application of the algorithm of Nagamochi and Ibaraki to generate G from G', does not increase the complexity of the checker: According to Lemma 2.1 in [53], every cut vertex and separation pair in G must also be a cut vertex and separation pair in G', respectively. As G

is a spanning subgraph of G', every certificate for the 3-connectivity of G can be augmented to a certificate for the 3-connectivity of G' by checking that G' differs from G only in additional edges.

3.6.1 Verifying Vertex and Edge Cuts

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 respective not k-edge-connected. However, as checkers must be very simple, we 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 has a red and a green end vertex.

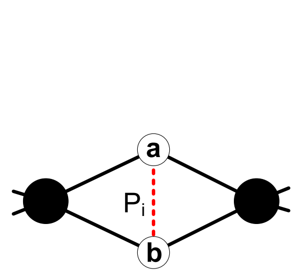
For G being not k-edge-connected, it suffices to check that there is at least one red, one green and no uncolored vertex and that the end vertices of at most $k-1$ edges differ in color. In particular, this certifies a graph to be disconnected for $k = 1$. We will always use these certificates instead of using X itself. The certificates can be easily computed from gives vertex and edge cuts by the certifying algorithm and need space $O(n)$. They can be checked in time $O(m)$, as the condition $n > m + 1$ can be checked in advance and, if true, proves G to be disconnected.

For certifying algorithms that test a graph G on being k-connected or k-edge-connected for $k \leq 3$, it remains to show how vertex and edge cuts X for the red-green coloring can be computed and which certificates are used if G is k-connected or k-edge-connected for $1 \leq k \leq 3$. Performing a depth-first search [43, 64, 67] or any other suited graph traversal gives the connected components of G in linear time. A certificate for these connected components and, thus, for the (1-)connectivity of G, is given in [44], using an easy numbering scheme on the vertices. For testing a graph on 2-connectivity and 2-edge-connectivity, we defer to Section 4.1.1. The next sections discuss certificates for 3-connectivity and 3-edge-connectivity.

3.6.2 Verifying 3-Connectivity

Let G be the input graph and let Q be a sequence (3.10) of G. We use Q in the path representation as certificate for 3-connectivity. A checker for this certificate has to check the following properties.

- Q is correct
- Q constructs G
- $\delta(G) \geq 3$ (according to Theorem 37)



(a) a and b are adjacent and either a or b has degree 2 (here $deg(b) = 2$)

(b) a and b are adjacent and $deg(a) = deg(b) = 2$

(c) $deg(a) = deg(b) = 2$ and $N(a) = N(b)$

Figure 3.15: Cases in which Properties 30b (see Figures (a) and (b)) and 30c (see Figure (c)) are violated before smoothing a and b.

We first check that every vertex in G has degree at least 3 in time $O(n)$. The path representation requires S_4 and all (BG-)paths $P_4, \ldots, P_{z-1}$ to be stored as subgraphs of G. It can be checked in $O(m)$ time that S_4 is a subgraph of G and that every P_i, $4 \leq i \leq z - 1$, is a path in G. To ensure that Q constructs G, it suffices to check that the paths P_i partition $E(G) \setminus E(S_4)$ in time $O(m)$. It remains to check that Q is correct.

We could validate this by transforming the path representation to the edge representation with Lemma 36 and checking the validity of each BG-operation by comparing labels, but this is too complicated for a checker.

Instead, we remove the paths P_i in reversed order from G. This is only well-defined if each P_i is an edge. If we encounter a path P_i that contains more than one edge immediately before deleting it (this can occur, e. g., when BG-paths are given in the wrong order), every inner vertex x of P_i must have degree 2 in the current subgraph, as otherwise P_i would violate Property 30a. However, this contradicts that x would have been deleted by a previous smoothing due to $deg(x) \geq 3$ in G. We conclude that Q can only be correct if every path P_i contains exactly one edge before removing it.

In that case the procedure passes through the graph sequence $G_z, \ldots, G_4$. It remains to verify that every removed edge $P_i = ab$ corresponds to a BG-path. We go along the Definition 30 of BG-paths (alternatively, the definition of BG-operations may be used). For Property 30a, it suffices to check that a and b are contained in the current subgraph.

Properties 30b and 30c can now be checked in constant time: Consider the situation immediately after the deletion of ab, but before smoothing a and b. Then all links in our subgraph are single edges, except possibly the ones containing a and b as inner vertices.

Therefore, 30b is not met for P_i if and only if a is adjacent to b and $(deg(a) = 2$ or $deg(b) = 2)$ (see Figures 3.15(a) and (b) for the two possible cases). Property 30c is not met if and only if $deg(a) = deg(b) = 2$ and $N(a) = N(b)$ (see Figure 3.15(c)). Both conditions can be checked in constant time. Note that encountering BG-paths $P_{z-1}, P_{z-2}, \dots, P_i$ does not necessarily imply that the current subgraph is 3-connected, since a path P_j, $j < i$, that is no BG-path might occur later.

It remains to validate that the graph after removing all BG-paths equals K_4. This can done in constant time by checking it on being simple and having exactly 4 vertices of degree three.

We can verify a sequence (3.8) in exactly the same way, except that we additionally check that at least one path P_i is given and the first path is an edge of the K_4.

Lemma 47. *The sequences* (3.8) *and* (3.10) *can be verified in time* $O(m)$.

From the algorithm of Theorem 46, the verification of vertex cuts in Section 3.6.1 and Lemma 47, we obtain a certifying algorithm.

Corollary 48. There is a certifying algorithm with running time $O(n^2)$ that tests a simple graph G on 3-connectivity.

3.6.3 3-Edge-Connectivity

Galil and Italiano [22] showed that the problem of testing a graph on k-edge-connectivity can be reduced to the problem of testing a slightly modified graph on k-connectivity. We want to certify this reduction for $k = 3$ in order to deduce a certifying algorithm for 3-edge-connectivity from every certifying algorithm for 3-connectivity, without increasing the asymptotic running time.

For $k = 3$, the reduction modifies the simple input graph G in linear time to a graph with $m + 3n$ vertices and $3m$ 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 $v_1, \dots, v_{deg(w)}$ be the arc-vertices that are incident to w. Then the edges $(v_1v_2, v_2v_3, \dots, v_{deg(w)}v_1)$ are added to G' if they do not already exist. 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 [22]. Moreover, every vertex cut of minimal size in G' contains only arc-vertices (Lemma 2.2 in [22]). We apply a certifying 3-connectivity test on G' to obtain a certifying 3-edge-connectivity test for G in the same time and space. If G' is not 3-connected, the test on 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 $G \backslash X$ as described in Section 3.6.1.

Otherwise, G' is 3-connected and we get a sequence (3.8) 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 verify 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 $v+1$ 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 in detecting additional vertices) in G and that this does not harm the 3-edge-connectivity of G. The certificate needs linear space and can be checked in time $O(m)$.

Corollary 49. There is a certifying algorithm that tests a simple graph G on 3-edge-connectivity in time $O(n^2)$.

Chapter 4

Certifying 3-Connectivity in Linear Time

At the end, we had something complete that made everything obvious, that made us realize how we should have attacked the problem. It takes a long time to realize the right form.

John E. Hopcroft [20]

We show that all construction sequences of the last chapter can be computed in optimal time $O(m)$. This gives linear-time certifying algorithms that test graphs on k-connectivity and k-edge-connectivity for $k \leq 3$.

Section 4.1 introduces a decomposition of the input graph that partitions the edge set of the graph into cycles and paths, called *chains*. On a high level view, chains provide a structure that will allow us to compute the next step of the construction efficiently. In Section 4.2, a certifying linear-time algorithm is given that computes a construction sequence if the input graph is 3-connected and a separation pair or cut vertex if the input graph is not 3-connected.

4.1 Chain Decompositions

We introduce a very simple decomposition of graphs into cycles and paths, which links DFS-trees, ear decompositions and open ear decompositions [42, 83] without needing to compute low-points in advance (see [34] for a definition of low-points). This decomposition does not only unify existing linear-time tests on 2-connectivity [12, 16, 18, 19, 21, 64, 66] and 2-edge-connectivity [65, 71, 74], 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.

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 $C = \{C_1, \dots, C_{|C|}\}$ 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 $w \rightarrow_{T'} r$ until a vertex x is found that is either r or already contained in a chain. The traversed subgraph $vw \cup (w \rightarrow_{T'} x)$ 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 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., $C_1 < \dots < C_{|C|}$. Processing the vertices in ascending DFI-order is not crucial; any pre-order on $V(T)$ can be used instead. Clearly, the decomposition into chains can be computed in time $O(n + m)$.

4.1.1 Testing 2-(Edge-)Connectivity

We show that the connectivity, 2-connectivity and 2-edge-connectivity of simple graphs G can be deduced from any chain decomposition of G by testing very simple conditions.

Lemma 50. *Let C be a chain decomposition of a simple graph G. Then G is connected if and only if $|C| = m - n + 1$.*

Proof. Let G be connected. Then G contains exactly $m - n + 1$ backedges. Since every chain in C contains exactly one backedge, $|C| = m - n + 1$ holds.

Let $|C| = m - n + 1$. Assume to the contrary that G is not connected and let $T_1, \dots, T_j$ with $j > 1$ be the trees in the DFS-forest T. For every T_i holds $|E(T_i)| = |V(T_i)| - 1$. Thus, $\sum_{1 \leq i \leq j} |E(T_i)| = \sum_{1 \leq i \leq j} |V(T_i)| - j = n - j$. It follows from $|C| = m - \sum_{1 \leq i \leq j} |E(T_i)|$ that $|C| = m - n + j$. This contradicts the assumption, because $j > 1$. □

Lemma 51. *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 that starts on a proper ancestor of v can enter $T(w)$, as this backedge would have been processed before in the chain decomposition, contradicting that C_i is a chain. By construction, $v \neq r$ and it follows that v is a cut vertex, which contradicts the assumption.

Now let every vertex in G have degree at least two and let C_1 be the only cycle in C. Then $n > 2$ holds and G must be connected, as every connected component of G contains a cycle. 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 (we say 2-*components*), where K_2 is regarded as 2-connected. 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 [23, 32], 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 a K_2, 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. □

Note that it is necessary for 2-connected and 2-edge-connected graphs that the minimum degree $\delta(G)$ of G is at least two, as otherwise G would either be disconnected or contain a cut vertex. Lemma 52 will cover this necessity implicitly.

Lemma 52. *Let C be a chain decomposition of a simple graph G with $n > 1$. Then G is 2-edge-connected if and only if $|C| = m - n + 1$ 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. With Lemma 50, $|C| = m - n + 1$. 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 $|C| = m - n + 1$ and let the chains in C partition $E(G)$. According to Lemma 50, G is connected and T must be a tree. By assumption, $n > 1$. Assume to the contrary that G is not 2-edge-connected. Then G must contain a bridge. Let $e = xy$ be that bridge with x being an ancestor of y in T. As no backedge enters $T(y)$, e cannot be contained in any chain of C. This contradicts the assumption that C partitions $E(G)$. □

With computing one chain decomposition of G, we can easily check whether G is disconnected, connected, 2-connected and 2-edge-connected in linear time by using Lemmas 50, 51 and 52. 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, C contains at least one cycle $C_i \neq C_1$, according to Lemma 51. Then the vertex with minimal DFI in every such cycle $C_i \neq C_1$ 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. Once a cut vertex or bridge is computed, we can use the red-green coloring of Section 3.6.1 as certificate; the same holds when G is disconnected.

If G is 2-connected and 2-edge-connected, C will be an open ear decomposition and an ear decomposition [83], respectively. It is well-known that these decompositions witness the 2-connectivity respective 2-edge-connectivity of a graph [42, 83]. Therefore, we can use the decomposition into chains in both cases 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.

4.2 A Certifying Algorithm in Linear Time

For convenience, we will assume that the input graph G is simple throughout this chapter. As shown in Section 2.1, vertex connectivity is neither dependent on parallel edges nor on self-loops. 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 tests a graph actually 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 (3.8) as certificate and show how to compute it in linear time. According to Theorem 41, this implies that every sequence (3.1)–(3.10) of a 3-connected graph can be computed in time $O(n+m)$.

4.2.1 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 3.6.1 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 $O(m)$ and obtain the chains $C_1 < \cdots < C_{|C|}$. Additionally, we compute the minimum degree $\delta(G)$ of G during 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, yielding 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 51, 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 53. *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 with Lemma 1, Lemma 52 implies that the chains in C partition $E(G)$. Since G is in particular connected, $|C| = m - n + 1$ holds with Lemma 50. If r would have more than one child, the DFS-tree would imply that r is a cut vertex, contradicting the 2-connectivity of G. □

4.2.2 Computing a K_2^3-Subdivision

Assume for a moment that G is 3-connected. According to Lemma 32, it suffices to add iteratively BG-paths to an arbitrary prescribed K_2^3-subdivision S_3 in G to get a construction sequence (3.8) 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 is that Lemma 32 can always be applied on the new generated subgraph and therefore ensures a completion of the sequence. With Lemma 42, $S_z = S_{m-n+2} = G$. 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 but r has exactly one child in T. Therefore, at least two chains exist and the chains C_1 and C_2 must both start at r. According to Lemma 51, 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 further explanations as simple as possible, we 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 $C_1 = r \to_{C_1 \setminus E(C_0)} t(C_2)$.

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 54. *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 52, every edge in G is contained in exactly one chain. We define parents and children of chains.

Definition 55. 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 56. *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 57. *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 4.2.3. 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 4.2.4).

To make the computation efficient, Section 4.2.5 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 4.2.6. Section 4.2.7 deals with the computation of the restricted construction sequence in linear time by a reduction to interval overlaps.

4.2.3 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 $<$. The types are defined 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 4.1, 4.2 and 4.9(b).

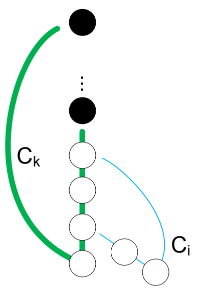
Algorithm 1 classify($C_i \in C \setminus \{C_0\}$, DFS-tree T)

```
 1: $C_k := parent(C_i)$                    ▷ the parent of $C_i$ in $U$: $C_k < C_i$
 2: if $t(C_i) \rightarrow_T s(C_i)$ 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: $C_k \neq C_0$, $t(C_i)$ 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: $s(C_i) \neq s(C_k)$, $C_k \neq C_0$, $t(C_i)$ 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 $L_i$; $C_j := parent(C_j)$
```

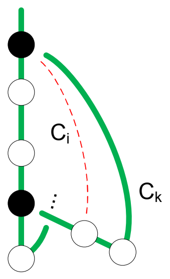
We first prove a basic property of chains of Types 2 and 3 and then show that the classification of chains can be carried out in linear time.

Lemma 58. *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.*

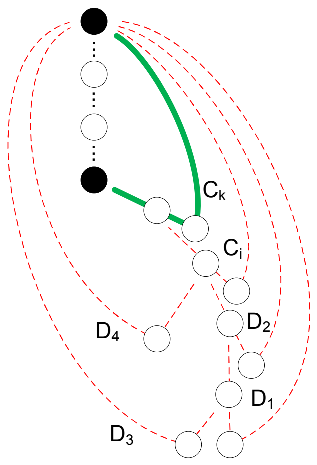
Proof. Assume to the contrary that $C_k = C_0$. Because $t(C_i)$ is contained in C_0, $s(C_i)$ must be in C_0 as well. But then C_i would be of Type 1, since $t(C_i) \rightarrow_T s(C_i) \subseteq C_0$.



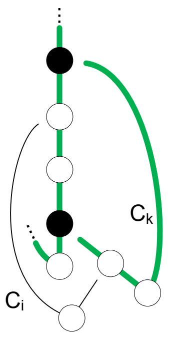
(a) C_i is of Type 1.



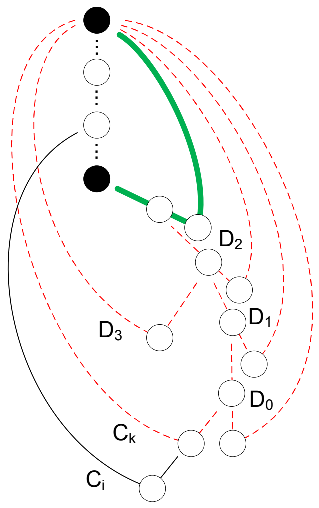
(b) C_i is of Type 2a, as it is a backedge.



(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 4.1: Different types of chains. Light solid (blue) chains are of Type 1, (red) dashed ones of Type 2 and black solid ones of Type 3.

Therefore, $C_k \neq C_0$ holds 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 59. *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. 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 [35] 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.

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 $t(C_i) \to_T s(C_i)$.

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

Lemma 60. *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 the claim. Every backedge that enters $T(x)$ must start at a vertex in $t(C_i) \to_T s(C_i)$, as otherwise either the DFS structure or the traversal of C_i by the chain decomposition would be violated. As Property B does not hold for C_i, 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 as 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 61. *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 only with Type 3 chains, but we omit the proof for the sake of a clearer presentation), we traverse the path $P = D_0 \rightarrow_U C_0$ 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 $t(C_i) \rightarrow_T s(C_i)$ 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 $D_0 \rightarrow_U C_i$. As e is the first traversed backedge entering $T(x)$ such that $s(e)$ is an inner vertex of $t(C_i) \rightarrow_T s(C_i)$, Lemma 56 implies $s(D_1) = s(D_2) = \dots = s(D_k) = s(C_i)$. 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.

Let C_i be a chain that has not 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 56 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 $t(C_i) \rightarrow_T s(C_i)$, $s(e)$ 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 61. If Property B is violated by a chain C_i, G has connectivity 2 due to Lemma 60 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 $<$ in that subset.

Lemma 62. *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 63. *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 53, 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 types of chains, C_j must be of Type 3. □

4.2.4 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 4.1(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 64. *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 65. *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 64, 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. We show that C_i is not marked anymore after all chains in C have been classified. This forces C_i to be contained in exactly one caterpillar, as the only way to unmark chains is to append them to a caterpillar (see Line 15 of Algorithm 1) and no chain is marked twice.

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 63, there is a chain of Type 3 that enters $T(x)$ and starts at an inner vertex in $t(C_i) \rightarrow_T s(C_i)$. 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 56 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 $V(C_j \rightarrow_U C_i) \setminus \{C_j\}$ 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 66. *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 $t(D_k) \rightarrow_T s(D_k)$.*

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 63, $s(C_i)$ is an inner vertex in $t(D_k) \rightarrow_T s(D_k)$. □

Definition 67. 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 68. A *cluster* is either a caterpillar or a chain of Type 1, 2a or 3a.

With Lemma 65, 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 69. 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 70. 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 $\prec$.

4.2.5 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 71. 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 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. In order to find BG-paths efficiently, we want to restrict every S_l to be upwards-closed and modular. However, this is not possible, as the following example shows. Consider $S_3 = \{C_0, C_1, C_2\}$ in the graph of Figure 4.2. As every BG-path for S_3 has end vertices x and y, S_4 cannot be modular.

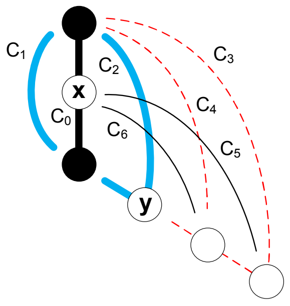
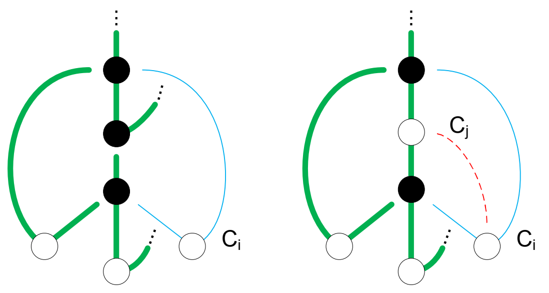


Figure 4.2: C_1 and C_2 are of Type 1, C_3 is of Type 2b, C_4 of Type 2a, C_5 of Type 3b and C_6 of Type 3a. No BG-path for the subgraph S_3 (depicted with thick blue edges) preserves modularity.

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.

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 will prevent BG-path candidates from violating Property 30c due to the DFS-structure (see Figure 4.3). 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 (3.8) 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.



(a) allowed (b) forbidden

Figure 4.3: The effect of Restriction (R_2) on adding the BG-path C_i to the fat subgraph. Note that adding C_i in Figure (b) would allow the chain C_j to violate Property 30c next.

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

(R_1) – can be decomposed into as many subsequent BG-paths as it contains chains and

– generates 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 53, $|C| - 3 = m - n - 1$, which is necessary for every sequence (3.8), as shown in Lemma 42.

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 4.2.6.

For simplicity, we say for a cluster that satisfies (R_1) and (R_2) on S_l^R that it *can be added*. We first show that Restriction (R_2) implies Property 30c.

Lemma 72. *Every path P for S_l^R with Properties 30a and 30b is a BG-path. If P is additionally a chain of Type 2a or 3a, P can be added.*

Proof. For the first claim, assume to the contrary that P violates Property 30c. 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 $S_l^R \neq S_3^R$.

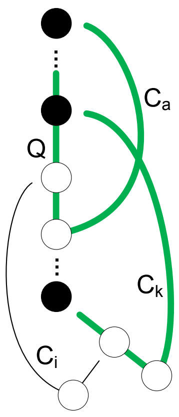


Figure 4.4: The chain C_i of Type 3 can be added to the fat subgraph.

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 54, 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 30a.

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 30b. 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 can be added to S_l^R if not already contained in S_l^R.

Lemma 73. *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 can be added.*

Proof. Since S_l^R is upwards-closed, modular and contains C_k, C_i satisfies Property 30a 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 30b. Lemma 72 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 can be added.

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 30b. Using Lemma 72, C_i can be added.

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 56, $s(C_i)$ must be an inner vertex of the path $t(C_k) \to_T s(C_k)$ (see Figure 4.4). 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 30b. Using Lemma 72, C_i can be added. □

According to Lemma 73, 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 yields 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 can be added. We next give similar conditions for the remaining Types 2b and 3b. According to Lemma 65, these chains are exactly the ones that are contained in caterpillars.

Definition 74. 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 4.5(a)). Otherwise, L_i is called a *good* caterpillar (see Figures 4.5(b) and (c)).

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 66, $s(C_i) \in V(t(C_k) \to_{C_k} y) \setminus \{y\}$. We characterize good caterpillars.

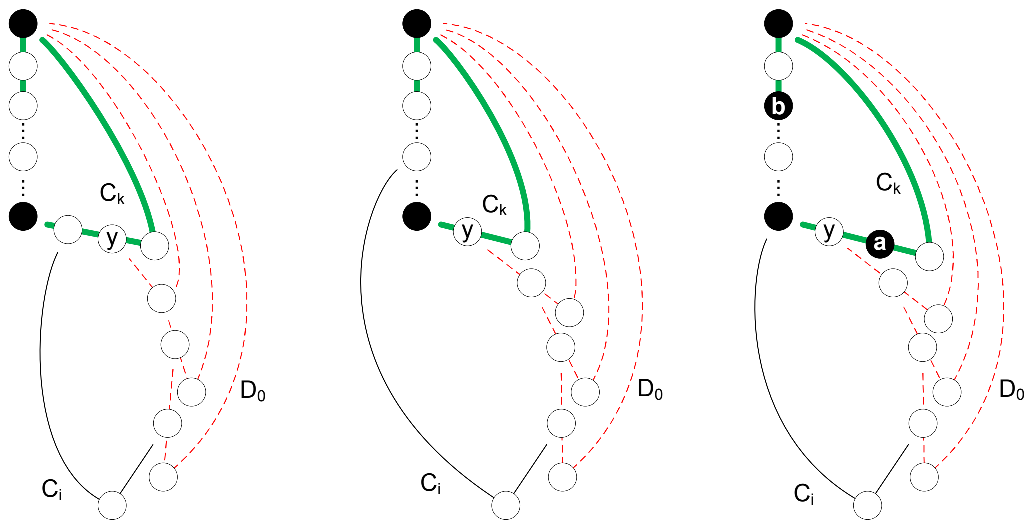
Lemma 75. *A caterpillar L_i with parent C_k is good if $s(C_i)$ is either an inner vertex of $t(C_k) \to_T s(C_k)$ (see Figure 4.5(b)) or a vertex in C_k such that $s(C_i) \to_{C_k} s(C_k)$ contains an inner real vertex (see Figure 4.5(c)).*

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 66. 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 can be added under minor assumptions.

Lemma 76. *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 can be added.*

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,



(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 $t(C_k) \rightarrow_T s(C_k)$.

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

Figure 4.5: Kinds of caterpillars.

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 $S_{l+1}, S_{l+2}, \ldots, S_{l+t}$ 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 $t(C_k) \rightarrow_t s(C_k)$ (see Figure 4.5(b)). Then the path $P = C_i \cup (t(C_i) \rightarrow_T y)$ fulfills Properties 30a and 30b and is a BG-path for S_l^R with Lemma 72. 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 $s(C_j) \rightarrow_{C_j} v$ 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 75 that $s(C_i)$ is contained in C_k with an inner real vertex a in $s(C_i) \rightarrow_{C_k} s(C_k)$ (see Figure 4.5(c)). We first show that $t(C_k) \rightarrow_T s(C_k)$ 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 $t(C_k) \rightarrow_T s(C_k)$ 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 $(C_i \cup D_0) \setminus ((t(C_i) \rightarrow_T y) \setminus t(C_i))$ 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 $t(C_i) \rightarrow_T y$, 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 72, 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 can be added. $\square$

Note that the decomposition of a good caterpillar L_i into subsequent BG-paths as shown in Lemma 76 can be computed in time linearly dependent on the edges in L_i.

4.2.6 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 77. 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 78. 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 [3], segments were used to design many efficient algorithms for problems related to planarity [14, 25, 34, 45, 50, 60, 80, 85] and 3-connectivity [35, 79, 80]. 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 63, we can deduce this result (in our notation) also for G, although G is not known to be 3-connected.

Lemma 79. *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 63, 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 80. *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.*

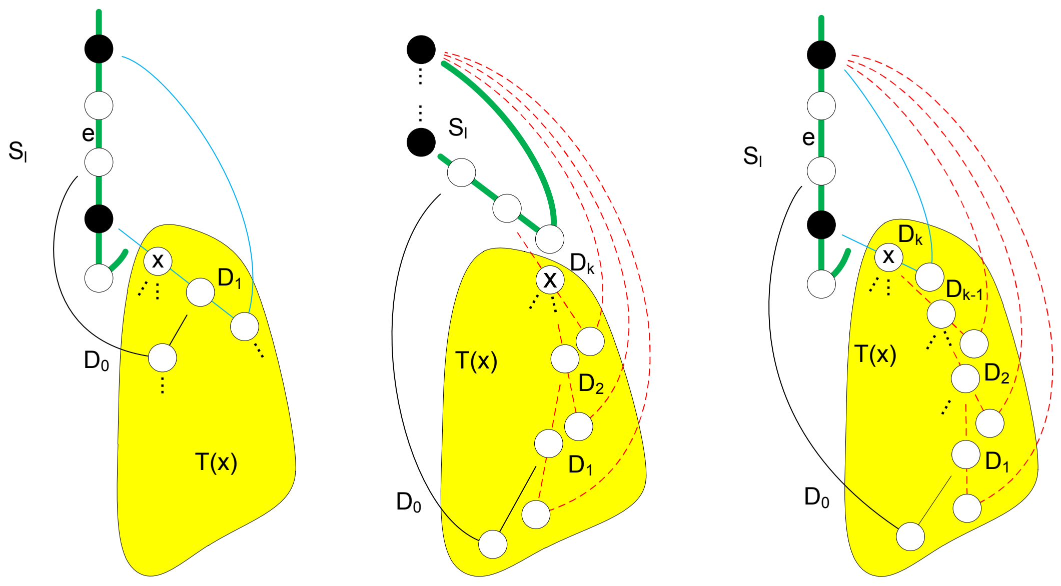
Proof. Recall that upwards-closedness and modularity of S_l^R implies that there is a subtree U' of U such that H is the union of the chains in U'. Clearly, only the root D_k of U' (i. e., the minimal chain in H) can be a child of a chain in S_l^R. The chain D_k is a child of a chain in S_l^R, as $t(D_k)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

Lemma 81. *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 $D_k, \ldots, D_0$ can be successively added to S_l^R, unless one of the following exceptions holds.*

1. *D_0 is of Type 3a, $k = 1$, D_k is of Type 1, $s(D_0)$ is an inner vertex of $t(D_k) \rightarrow_T s(D_k)$ and there is no inner real vertex in $t(D_k) \rightarrow_T s(D_k)$ (see Figure 4.6(a)),*
2. *D_0 is of Type 3b and $\{D_0, \ldots, D_k\}$ is a bad caterpillar (see Figure 4.6(b)),*
3. *D_0 is of Type 3b, $\{D_0, \ldots, D_{k-1}\}$ is a caterpillar, D_k is of Type 1, $s(D_0)$ is an inner vertex of $t(D_k) \rightarrow_T s(D_k)$ and there is no inner real vertex in $t(D_k) \rightarrow_T s(D_k)$ (see Figure 4.6(c)).*

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



(a) Exception 81.1 (b) Exception 81.2 (c) Exception 81.3

Figure 4.6: The three exceptions of Lemma 81. The black vertices in Exceptions 81.1 and 81.3 may also be non-real.

Let D_0 be of Type 3a. If $k = 0$, $t(D_0)$ is contained in S_l^R and D_0 can be added with Lemma 73, 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 4.6(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 can be subsequently added with Lemma 73. Otherwise, Exception 81.1 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 can be added with Lemma 76. If $t = k$ and L_i is bad, Exception 81.2 holds (see Figure 4.6(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 4.6(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 63 to the chain D_{t+1} yields 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 81.3 holds. Otherwise, Lemma 73 implies that D_k can be added. After adding D_k, L_i is good due to Lemma 75 and can be added with Lemma 76. □

We extend Lemma 81 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 82. *Let the preconditions of Lemma 81 hold and let D_0 be not contained in one of the Exceptions 81.1–81.3. 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 can be successively added 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 can be successively added by iteratively applying Lemma 81. 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 81). By definition of Y, the leaves of U_Y are chains of Type 3 that start at a vertex in S_l^R.

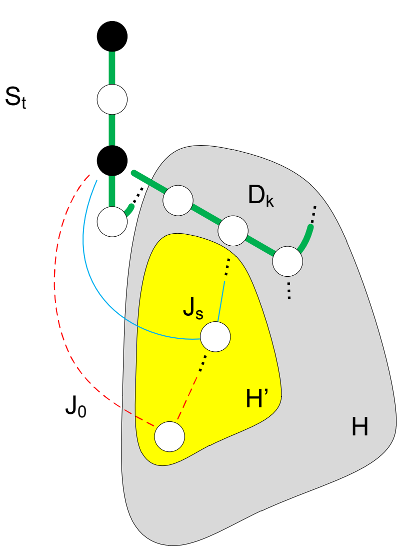
Let any pre-order σ on the clusters of Y be given that adds clusters that start at $t(D_k) \in V(S_l^R)$ last. We go along σ. After adding an arbitrary number of clusters in the order of σ, let J be the next cluster to add. Let J_s be the minimal chain in J and let S_t^R be the current subgraph of G. 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 $J_s < \dots < J_0$ be all ancestors of J_0 in U'.

We apply Lemma 81 to J_0 in H' and show that J_0 cannot be contained in one of the Exceptions 81.1–81.3. This implies that the clusters of $J_s, \dots, J_0$ can be added in pre-order. The claim follows from merely adding J and iterating the argument for the next cluster in σ.

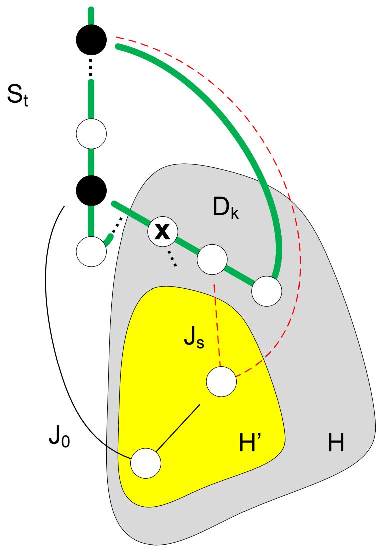
It remains to show that J_0 is not contained in one of the Exceptions 81.1–81.3. By assumption, this holds for $J_0 = D_0$; in that case, the cluster of D_k is added. As σ is a pre-order, the cluster of D_k is the first cluster in H that is added. We can therefore assume for the following arguments that D_k is in S_t^R.

First, assume to the contrary that J_0 is contained in Exception 81.1 or 81.3 (see Figure 4.7(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 80, 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 56. This contradicts J_0 to be in Exception 81.1 or 81.3, because $s(J_0)$ is not an inner vertex of $t(J_s) \rightarrow_T s(J_s)$.



(a) J_0 is not contained in Exceptions 81.1 and 81.3.



(b) J_0 is not contained in Exception 81.2.

Figure 4.7: In S_t^R, J_0 is not contained in one of the Exceptions 81.1–81.3.

Now assume to the contrary that J_0 is contained in Exception 81.2 (see Figure 4.7(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 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 80, 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 last but one vertex of D_k. Applying Lemma 63 on D_k yields that a chain C_y of Type 3 enters $T(x)$ with $s(C_y)$ being an inner vertex of $t(D_k) \rightarrow_T s(D_k)$. 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 83. 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 be added 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 their clusters will be added before in the course of adding the clusters of $\mathit{Type}_3(C_j)$ for an ancestor C_j of C_i.

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

Lemma 84. *Let C_i be a chain in S_l^R such that* $\mathrm{Type}_3(C_j) = \emptyset$ *holds for every proper ancestor C_j of C_i. Let B be the subset of chains in* $\mathrm{Type}_3(C_i)$ *whose segments do not contain a chain in* $\mathrm{Children}_{12}(C_i)$*. Then the clusters of all ancestors of chains in B that are not in S_l^R can be successively added. 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 $t(C_i) \to_{C_i} s(C_i)$, 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 $B \cap H$ can be added. 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 56, $s(D_0)$ 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 82 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 82, it suffices to show that D_0 is not contained in one of the Exceptions 81.1–81.3. 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 81.1–81.3.

First, assume to the contrary that D_0 is contained in Exception 81.1 or 81.3. Then D_k is of Type 1. It remains 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, $t(D_k) \to_T s(D_k)$ must be contained in the parent of D_k. In both Exceptions, $s(D_0)$ is an inner vertex of $t(D_k) \to_T s(D_k)$ that is additionally non-real, as $t(D_k) \to_T s(D_k)$ 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 81.2. 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 $H \cap \mathit{Children}_{12}(C_i) = \emptyset$.

Assume to the contrary that $D \neq C_i$. As $\{D_0, \ldots, D_k\}$ 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 $\{D_0, \ldots, D_k\}$ 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 $t(D) \to_T s(D)$ due to Lemma 63. 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 81.1–81.3 in Lemma 81, 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 85. 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 can be added. 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 73 and 76. In the case when the parent of D is contained in S_l^R but P is not empty, the fact whether D can be added will be essentially dependent 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 86. *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 56, 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 of which every chain 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 56 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 $P = s(C_i) \rightarrow_{C_k} s(C_k)$. □

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

Theorem 87. *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 can be successively added.*

Proof. By applying Lemma 84 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 80, D_k is 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 can be added. 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 84. 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 $\mathit{Children}_{12}(C_i)$ cannot be added. 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 80. Let D be of Type 1. Then the dependent path P of D does not contain an inner real vertex, as otherwise D can be added with Lemma 73. According to Lemma 63, 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 82 and it follows that D must be contained in Exception 81.1 or 81.3 (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 can be added due to Lemma 73. 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 cannot be added, this caterpillar must be bad with Lemma 76 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 82 that D is contained in Exception 81.2 (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 81.1 or 81.3
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 81.2

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 79. 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 32 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 63 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 63 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 86, this holds in particular for v. As v is non-real and H is not contained in S_l^R, B must contradict Property 30b. 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 $t(C_a) \rightarrow_T v$ 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 that the claim. □

Assume for a moment that G is 3-connected. We show that every S_l^R contains a chain C_i that satisfies the preconditions of Theorem 87, starting with C_0 in S_3^R. Applying Theorem 87 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 87 on $C_0, C_1, \ldots, C_{m-n+1}$ constructs G, which proves the existence of the restricted construction sequence.

Corollary 88. 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 $S_3^R = \{C_0 \cup C_1 \cup C_2\}$.

4.2.7 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 4.2.1–4.2.3).

Although G is not known to be 3-connected, the proof of Theorem 87 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 can be added. This order does not necessarily exist, as G might be not 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 $(t(C_i) \rightarrow_{C_i} s(C_i)) \setminus s(C_i)$.

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 73.

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 get D by stopping at the first vertex that is marked. 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 80, 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 84, the clusters of all ancestors of the chains in X that are not in S_l^R can be added successively. Moreover, these clusters can be added 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 84 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 80. According to Theorem 87, 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 can be successively added if G is 3-connected. However, Theorem 87 does not specify in which order these clusters can be added, so we need to compute a valid order on them (if this order exists).

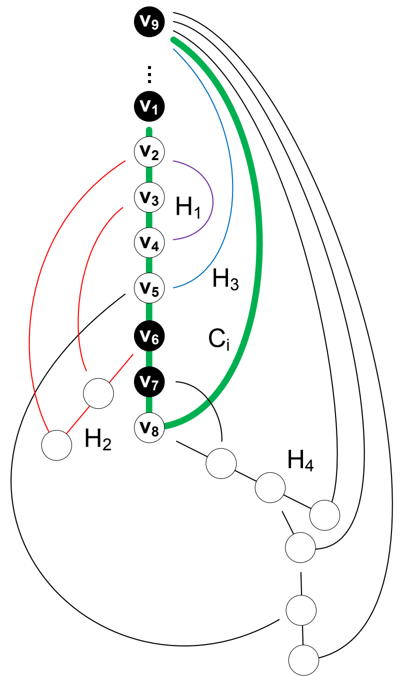
Let H be the segment of a chain D in $\mathit{Children}_{12}(C_i)$. The cluster of D cannot be a caterpillar L_j with $s(C_j) \notin V(C_i)$, as we maintain the invariant that $\mathit{Children}_{12}(C_k) = \mathit{Type}_3(C_k) = \emptyset$ for every proper ancestor C_k of C_i. Thus, D is dependent on a non-empty path $P \subseteq C_i$ by Definition 85. This path contains all attachment vertices of H with Lemma 86. 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 can be added, 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 can be added if and only if P contains an inner real vertex.

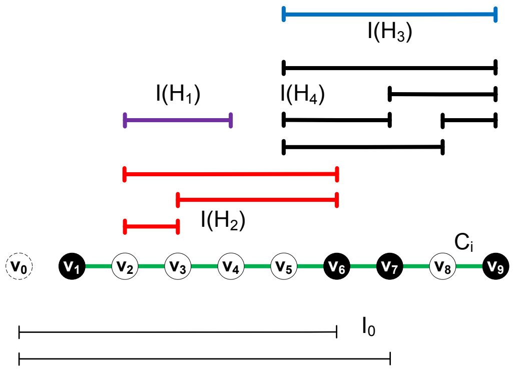
Lemma 89. *Let D be a remaining chain in* $\mathrm{Children}_{12}(C_i)$ *in S_t^R, let H be the segment of S_t^R that contains D and let P be the dependent path of D. If P contains an inner real vertex, the clusters of all ancestors of the chains in* $H \cap (\mathrm{Children}_{12}(C_i) \cup \mathrm{Type}_3(C_i))$ *that are not in S_t^R can be successively added. Moreover, these clusters can be added 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 can be added.*

Proof. Let P contain an inner real vertex. If $H \cap \mathit{Type}_3(C_i) \neq \emptyset$, the claim follows directly from Lemma 82, as the dependent path of the minimal chain (D_k) of each of the Exceptions 81.1–81.3 contains no inner real vertex. If $H \cap \mathit{Type}_3(C_i) = \emptyset$, we only need to show that the cluster of D can be added. Moreover, D must be of Type 1 or 2a, as otherwise $H \cap \mathit{Type}_3(C_i) \neq \emptyset$. According to Lemma 73, D can be added.

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



(a) The chain C_i and the remaining chains in $Type_3(C_i) \cup Children_{12}(C_i)$.



(b) The intervals in I_0 are constructed from the two inner real vertices v_6 and v_7 in C_i.

Figure 4.8: Mapping segments to intervals. Different colors depict different segments.

adding each cluster, the cluster of D must be added first. According to Lemma 86, 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 30b. Since $D \in Children_{12}(C_i)$, 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 73.

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 $Type_3(C_k) = \emptyset$ 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 30a–c. This contradicts Restriction (R_1), as L_j, the cluster of D, can not be decomposed into BG-paths. □

4.2.7.1 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 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 89, finding an order in which the clusters of B can be added 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 [79, 80] and based on the work of Williamson [84]. 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 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 $a_1, \dots, a_k$ 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 4.8). Additionally, we augment $V(C_i)$ by an artificial first vertex v_0 and map the real vertices $b_1, \dots, b_k$ 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 $|\mathit{Children}_{12}(C_i)| + 2|\mathit{Type}_3(C_i)| + |V_{real}(C_i)| - 2$ 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 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 90. *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 $H_0, H_1, \dots, H_{|Y|}$ 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 $\sigma = H_1, \dots, H_{|Y|}$ is a proper order on Y. Let H_i, $1 \le i \le |Y|$, 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 $\sigma = H_1, \dots, H_{|Y|}$ be a proper order on Y. We show that every H_i, $1 \le i \le |Y|$, 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 $1 \le j < i$, 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 $1 \le k < j$, 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 87 and Lemma 89. The merged overlap graph G' of Y is then connected with Lemma 90. The following algorithm detects whether G' is connected and computes σ in the affirmative case and the connected components of G' otherwise.

Lemma 91. *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 90. 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 [54] 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 [54] describes mainly a non-sequential variant of the algorithm; for a simple sequential variant, Lemmas 4.1 and 4.2 in [54] 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 Lemma 90, 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 $|Children_{12}(C_i)| + 2|Type_3(C_i)| + |V_{real}(C_i)| - 2$ intervals are created. Thus, applying the algorithm of Lemma 91 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' exist. According to Theorem 87 and Lemma 90, 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 92. *The sequences* (3.1)–(3.10) *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.

4.2.7.2 Extracting a Separation Pair

We show how a separation pair can be extracted if not all clusters of $Children_{12}(C_i) \cup Type_3(C_i)$ have been added after processing C_i. Then $Children_{12}(C_i)$ 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 91.

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 89. 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 79.

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 93. *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* (3.1)–(3.10) *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 4.9. We get the following corollary from the discussion of certificates for 3-edge-connectivity (see Section 3.6.3) and edge cuts (see Section 3.6.1).

Corollary 94. There is a certifying algorithm that tests the 3-edge-connectivity of a simple graph G in time $O(n+m)$.

4.2.7.3 Simplifications

We list some simplifications for Algorithm 2.

- 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 $deg(v) \geq 3$ 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.

Algorithm 2 certify(Graph G)

```
1: Compute a DFS-tree $T$ of $G$, a chain decomposition $C$ and classify the chains
2: Check Properties A and B, Compute $S_3^R := C_0 \cup C_1 \cup C_2$ ▷ Sections 4.2.1–4.2.3
3: for $i := 0$ to $m - n + 1$ do                                  ▷ process each $C_i$
4:     Compute the lists $Children_{12}(C_i)$ and $Type_3(C_i)$              ▷ page 72
5:     for each $D \in Children_{12}(C_i)$ that is of Type 2a such that $t(D)$ is real do
6:         Add $D$; update $Children_{12}(C_i)$           ▷ optional, see Section 4.2.7.3
7:     Partition $Type_3(C_i)$ into segments                          ▷ page 72
8:     (every such segment is represented by the minimal chain it contains)
9:     for each segment $H$ with $H \cap Type_3(C_i) \neq \emptyset$ and $H \cap Children_{12}(C_i) = \emptyset$ do
10:        Add the clusters of all ancestors of $H \cap Type_3(C_i)$ that are in $H$ in the
11:        order of $<$; update $Type_3(C_i)$                            ▷ Lemma 84
12:    Let $Y$ be the set of segments that contain a chain in $Children_{12}(C_i)$
13:    for each $H \in Y$ do
14:        Compute the attachment vertices and the dependent path of $H$ ▷ page 73
15:        Map $H$ to a set of intervals on $C_i$                      ▷ Section 4.2.7.1
16:    Using these intervals, compute either a proper order $\sigma$ on $Y$ or more than
17:    one connected component of the merged overlap graph $G'$ of $Y$ ▷ Lemma 91
18:    if a proper order $\sigma$ was computed then
19:        for each segment $H \in Y$ in the order of $\sigma$ do          ▷ Add clusters
20:            Add the clusters of all ancestors of $H \cap (Type_3(C_i) \cup Children_{12}(C_i))$
21:            that are in $H$ in the order of $<$; update $Type_3(C_i)$     ▷ Lemma 89
22:    else                                                ▷ $G'$ is not 3-connected
23:        Compute a separation pair                               ▷ Section 4.2.7.2
```

- 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 $C_0, C_1, \ldots, C_{m-n+1}$.

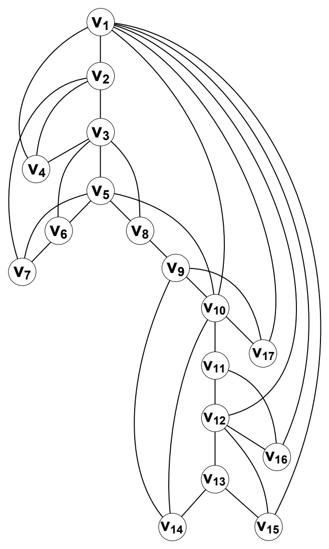
4.3 Conclusions

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 achieves optimal linear running 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 $O(n + m)$.

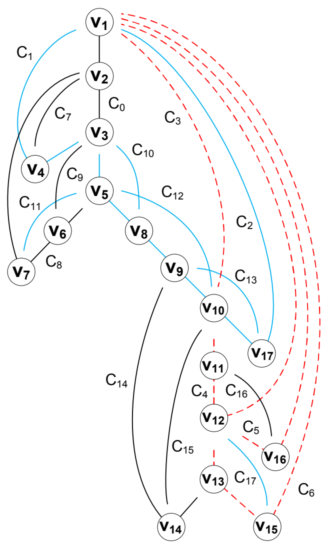
An $SPQR$-tree of a 2-connected graph G is a tree that represents the 3-connected components of G. Hopcroft and Tarjan [35] and Gutwenger and Mutzel [27] show that the $SPQR$-tree of a 2-connected graph can be computed in linear time. Clearly, we can certify the 3-connected components (the so-called R-nodes) of this tree to be 3-connected with the algorithm of Chapter 4. However, it would be interesting whether the algorithm itself can be extended to give such an $SPQR$-tree of the input graph.

In contrast to the non-certifying algorithms in [35, 79, 80], the algorithm given in this thesis does neither assume the graph to be 2-connected nor needs to compute low-points (as defined in [35]) 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.

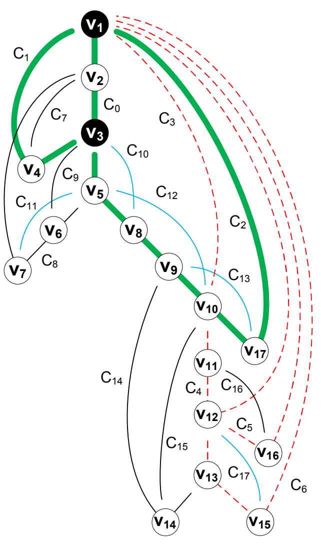
Another open question is whether the approach with construction sequences provides a way to test graphs efficiently on higher vertex connectivity. Up to now, no linear-time algorithm for testing graphs on 4-connectivity is known.



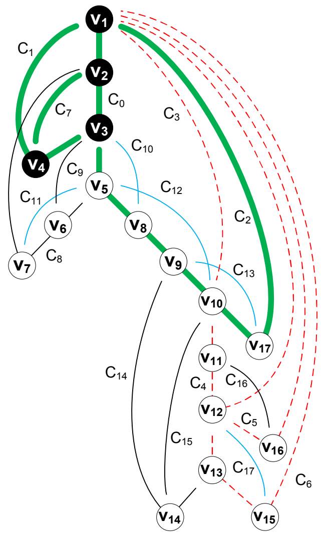
(a) A 3-connected input graph G with $n = 17$ and $m = 33$. Straight lines depict the edges of the DFS-tree T.



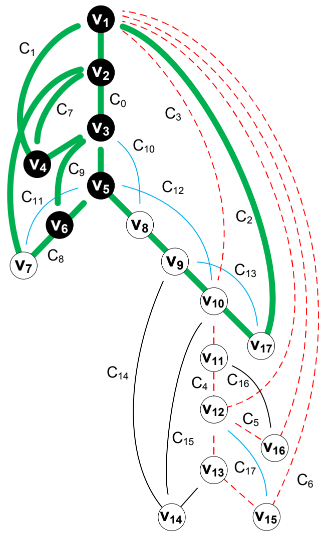
(b) The decomposition of G into $m - n + 2$ chains. Light solid chains are of *Type 1*, red dashed ones of *Type 2* and black solid ones of *Type 3*. The only chain of Type 2a is C_3. The only chains of Type 3b are C_{14} and C_{16}, giving the caterpillars $L_{14} = \{C_{14}, C_6, C_4\}$ and $L_{16}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



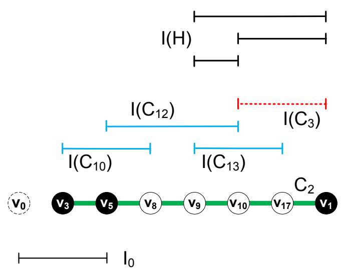
(c) The K_2^3-subdivision $S_3^R = \{C_0, C_1, C_2\}$ in G (thick green edges). We start with processing C_0. Since $Children_{12}(C_0) = \emptyset$, we can add all chains in $Type_3(C_0) = \{C_7, C_8, C_9\}$ with Lemma 84 in ascending order of $<$. Thus, we add C_7 first.



(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.

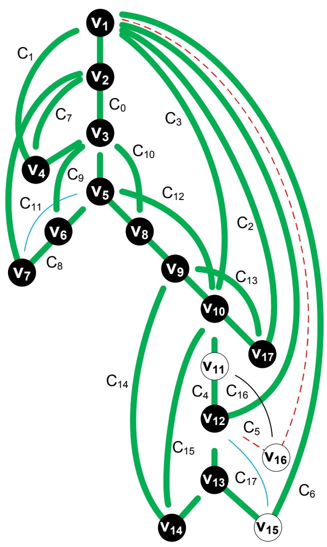


(e) The subgraph S_6^R. As there is nothing to process for C_1, we process C_2 next. Due to Theorem 87, C_3, C_{10}, C_{12}, C_{13}, C_{15} and L_{14} can be added if G is 3-connected. To obtain the right order on these clusters, we group them by segments and compute the following mapping.

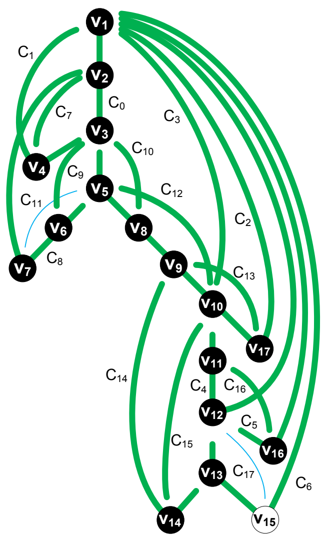


(f) Mapping segments to intervals (I_0 is induced by v_5). Let H be the segment of C_{14}. Any sequence of overlapping intervals, e. g., I_0, $I(C_{10})$, $I(C_{12})$, $I(C_{13})$, $I(C_3)$, $I(H)$, gives an order on segments. For each such segment H', we add the clusters of all ancestors of $H' \cap (Children_{12}(C_2) \cup Type_3(C_2))$ in H'.

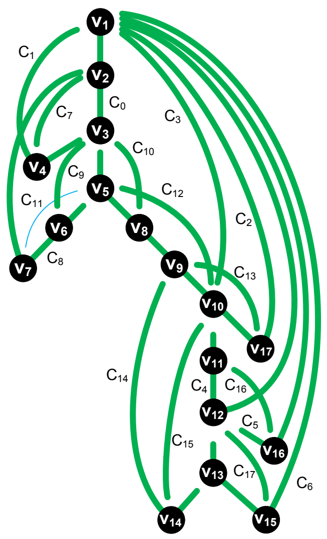
Figure 4.9: An example of the algorithm.



(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 89. The next non-trivial chain to process is C_4 with $Children_{12}(C_4) = \{C_5\}$ and $Type_3(C_4) = \{C_{16}\}$. 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 76.



(h) The subgraph S^R_{16}. As there is nothing to do for C_5, we next process C_6 with $Children_{12}(C_6) = \{C_{17}\}$ and $Type_3(C_6) = \emptyset$. 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} can be added.



(i) The subgraph S^R_{17}. The next non-trivial chain to process is C_8 with $Children_{12}(C_8) = \{C_{11}\}$ and $Type_3(C_8) = \emptyset$. 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} can be added as the last BG-path of the construction sequence. This generates the subgraph S^R_{18}, which is identical to G.

Figure 4.9: An example of the algorithm (continued).

Chapter 5

Contractible Edges in 3-Connected Graphs

The moving power of
mathematical invention is not
reasoning, but imagination.

Augustus De Morgan

All results of this chapter are joint-work with *Amr Elmasry* and *Kurt Mehlhorn* from Max-Planck-Institut für Informatik in Saarbrücken, Germany.

5.1 Introduction

Over 40 years ago, Tutte [76] proved the fundamental result that every 3-connected graph on more than 4 vertices contains a *contractible* edge, i. e., an edge $e = xy$ with $x \neq y$ that generates a 3-connected graph upon contraction. Since then, the distribution of contractible edges in 3-connected graphs has been intensively studied. Many papers establish lower bounds on the number of contractible edges [2, 55], or on entire contractible subgraphs [41]. See [40] for an excellent survey. Analogously, bounds on the number of removable edges in 3-connected graphs have been proved [33, 37].

We strengthen Tutte's result by showing that every depth-first search tree of a 3-connected graph contains a contractible edge. One might hope for more than one contractible edge in a depth-first search tree or for a generalization of this result to more general subgraphs, e. g., for spanning trees. However, we exhibit 3-connected graphs with a depth-first search tree containing exactly one contractible edge and 3-connected graphs with a spanning tree containing no contractible edge. We call a 3-connected graph a *fox* if it has a spanning tree containing no contractible edge. We present infinite families of foxes and give conditions under which a 3-connected graph is not a fox (see Lemma 104).

The results of this chapter are not only of structural interest. They may as well lead to new inductive proof methods for 3-connected graphs and to simpler certifying algorithms for testing 3-connectivity. As a first step in this direction, [17] derive a certifying algorithm in linear-time for the 3-connectivity of Hamiltonian graphs from the main result of this chapter (for the case that a Hamiltonian cycle of the input graph is part of the input).

5.2 Preliminaries

For a graph G and a vertex cut V' of G, the connected components of $G \setminus V'$ are called the *split components* (or *separation classes*) with respect to V'. Recall that vertex cuts of size one, two and three are called *cut vertices*, *separation pairs* and *separation triples*, respectively.

We use the following known results in our proofs.

Proposition 95. *An edge xy in a 3-connected graph on more than 4 vertices is contractible if and only if no separation triple containing x and y exists.*

Theorem 96 (Tutte [76], see also Corollary 10)**.** *Every 3-connected graph on more than 4 vertices contains a contractible edge.*

Lemma 97 (Halin [30])**.** *In a 3-connected graph on more than 4 vertices, every vertex of degree 3 has an incident contractible edge.*

Lemma 98 (Ota [55])**.** *Let v be a vertex of degree 3 in a 3-connected graph G on more than 4 vertices and let x, y, and z be its neighbors. If $xy \in E(G)$, then vz is contractible.*

Our main result is the following.

Theorem. *Let G be a graph on more than 4 vertices. Every depth-first search tree of G contains a contractible edge.*

5.3 Separation Triples and Split Components

We establish some useful properties of separation triples and split components.

Lemma 99. *Let $ST = \{x, y, z\}$ be a separation triple in a 3-connected graph G. Let D be one of the split components of $G \setminus ST$. Then, every vertex in ST has a neighbor in D.*

Proof. Assume otherwise, say z has no neighbor in D. Then, D is a split component of $G \setminus \{x, y\}$, a contradiction to G being 3-connected. □

Lemma 100. *Let $ST = \{x, y, z\}$ be a separation triple in a 3-connected graph G, and let D be one of the split components of $G \setminus ST$. If $ST' = \{x', y', z'\}$ is a separation triple in G with $ST' \neq ST$ and $ST' \subset V(D) \cup ST$, then there is a split component of $G \setminus ST'$ properly contained in D.*

Proof. Let $D, D_1, \ldots, D_j$ be the split components of $G \setminus ST$. Consider the components D_i, where $1 \leq i \leq j$. Every D_i is connected in $G \setminus (V(D) \cup ST)$, and hence connected in $G \setminus ST'$. Moreover, according to Lemma 99, any vertex in $ST \setminus ST'$ has a neighbor in D_i. It follows that $(ST \setminus ST') \cup V(D_i)$ is contained in a split component of $G \setminus ST'$. Since $ST \setminus ST'$ is non-empty, $(ST \setminus ST') \cup \bigcup_{1 \leq i \leq j} V(D_i)$ is contained in a split component of $G \setminus ST'$. Any other split component of $G \setminus ST'$ (there must be at least one) is contained in $V(G) \setminus ((ST \setminus ST') \cup \bigcup_{1 \leq i \leq j} V(D_i) \cup ST')$, and therefore properly contained in D. □

Lemma 101. *Let G be a 3-connected graph and let $\{x, y, z\}$ and $\{v, y, w\}$ be two separation triples in G intersecting exactly in y. Then, v and w are contained in the same split component of $G \setminus \{x, y, z\}$ if and only if x and z are contained in the same split component of $G \setminus \{v, y, w\}$. Moreover, if v and w belong to distinct split components, then each of $G \setminus \{x, y, z\}$ and $G \setminus \{v, y, w\}$ has exactly two split components.*

Proof. Assume that v and w are contained in the same split component of $G \setminus \{x, y, z\}$. Then, there is a split component S of $G \setminus \{x, y, z\}$ that contains a neighbor of x and a neighbor of z, but neither v nor w. As $S \cup \{x, z\}$ is connected in $G \setminus \{v, y, w\}$, x and z belong to the same split component of $G \setminus \{v, y, w\}$. Conversely, if x and z belong to the same split component of $G \setminus \{v, y, w\}$, then v and w belong to the same split component of $G \setminus \{x, y, z\}$ for the same reason. This proves the first claim.

Assume that there are more than two split components of $G \setminus \{x, y, z\}$. Then, among these split components there is a component containing neither v nor w. It follows that x and z belong to the same split component of $G \setminus \{v, y, w\}$; in accordance, v and w belong to the same split component of $G \setminus \{x, y, z\}$. The same arguments apply if there are more than two split components of $G \setminus \{v, y, w\}$. □

We call two separation triples $\{x, y, z\}$ and $\{v, y, w\}$ *crossing* if they intersect in exactly one vertex and if v and w belong to distinct components of $G \setminus \{x, y, z\}$. Then, x and z belong to distinct components of $G \setminus \{v, y, w\}$ by Lemma 101. In addition, both, $G \setminus \{x, y, z\}$ and $G \setminus \{v, y, w\}$, have exactly two split components.

Lemma 102. *Let G be a 3-connected graph, let $\{x, y, z\}$ and $\{v, y, w\}$ be two crossing separation triples in G, let D be the split component of $G \setminus \{x, y, z\}$ containing v and let X and Z be the split components of $G \setminus \{v, y, w\}$ containing x and z, respectively. Then, either $X \cap D = \emptyset$ or $\{x, y, v\}$ is a separation triple. Additionally, either $Z \cap D = \emptyset$ or $\{z, y, v\}$ is a separation triple.*

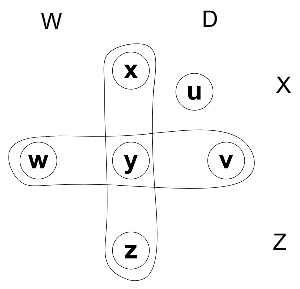


Figure 5.1: Two crossing separation triples.

Proof. Assume $X \cap D \neq \emptyset$. Consider any edge $ur \in E(G)$ with $u \in V(X \cap D)$ and $r \notin V(X \cap D)$. See Figure 5.1. Then, $r \in \{x, y, z, v, w\}$. However, $r \neq z$, because $\{v, y, w\}$ separates X from z, and $r \neq w$, because $\{x, y, z\}$ separates D from w. It follows that $\{x, y, v\}$ separates $X \cap D$ from the rest of G. Analogously, if $Z \cap D \neq \emptyset$, then $\{z, y, v\}$ separates $Z \cap D$ from the rest of G. □

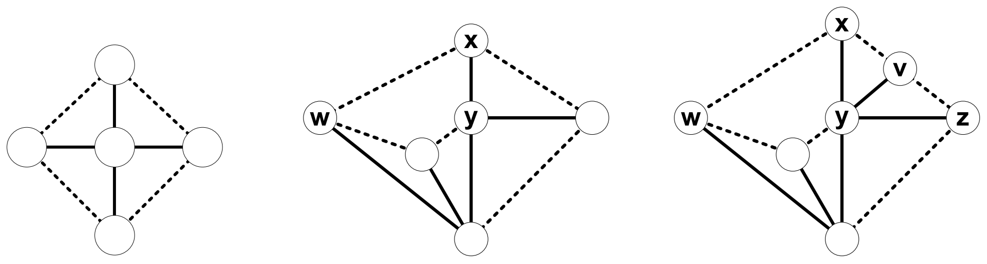
Lemma 103. *Let G be a 3-connected graph, let $ST = \{v, y, w\}$ be a separation triple in G and let X be a split component of $G \setminus ST$. If $G \setminus X$ is not 2-connected and b is a cut vertex of $G \setminus X$, then $b \notin ST$ and one of the vertices in ST has b as its only neighbor in $G \setminus X$ (and hence is a split component of $G \setminus (V(X) \cup \{b\})$). Conversely, if each vertex in ST has at least two neighbors in $G \setminus X$, then $G \setminus X$ is 2-connected.*

Proof. Assume that $G \setminus X$ is not 2-connected. Then, there is a cut vertex b that splits $G \setminus X$. If one of the split components of $G \setminus (V(X) \cup \{b\})$ does not contain a vertex from ST, then b is a cut vertex in G, which contradicts G to be 3-connected. It follows that every split component of $G \setminus (V(X) \cup \{b\})$ contains at least one vertex from ST. If $b \in ST$, say $b = y$, then $G \setminus (V(X) \cup \{b\})$ has exactly two split components; one containing v and one containing w. Since ST is a separation triple in G, there are vertices in $G \setminus X$ other than those in ST. It follows that one of the components of $G \setminus (V(X) \cup \{b\})$ must contain at least two vertices, say the component that contains w. Then, $\{y, w\}$ splits G, which contradicts G to be 3-connected. Therefore, $b \notin ST$.

The vertices of ST cannot all lie in one split component of $G \setminus (V(X) \cup \{b\})$. Hence, at least one of these split components, say S, contains exactly one vertex from ST, say w. If w has a neighbor in $G \setminus X$ other than b, then $|V(S)| > 1$ and $S \setminus w$ is a split component of $G \setminus \{b, w\}$, which contradicts G to be 3-connected. □

5.4 Contractible Edges and Spanning Trees

We next give a sufficient condition for every spanning tree of a 3-connected graph to contain a contractible edge.



(a) W_5 (b) $n = 6$ (c) $n = 7$

Figure 5.2: The solid edges are non-contractible and form a spanning tree.

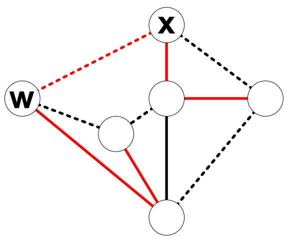


Figure 5.3: A depth-first search tree (depicted in red) that contains only one contractible edge, namely wx.

Lemma 104. *Let G be a 3-connected graph on more than 4 vertices and let F be an edge cut of G. If every edge e in F has an end vertex x, where $\deg(x) = 3$ and x has two neighbors in $G \setminus F$ adjacent to each other, then G is not a fox.*

Proof. Let v and w be the two neighbors of x in $G \setminus F$. Since v and w are adjacent, Lemma 98 implies that e is contractible. Therefore, every edge in F is contractible, and hence every spanning tree of G contains at least one contractible edge. □

Examples: There are arbitrary large foxes; the wheel graphs W_n, $n \geq 5$, with the spokes as the spanning tree form an infinite family, see Figure 5.2(a). Figure 5.2(b) shows the base graph of another infinite family of examples. In this graph, the vertices x, y, and w play a special role. The next larger graph in this family is generated as follows: Let v be the neighbor of x that is neither y nor w in the smaller graph, subdivide xv by one vertex and connect the new vertex with y; see Figure 5.2(c).

We will show that every depth-first search tree of a 3-connected graph contains a contractible edge. The graph on 6 vertices of Figure 5.3 illustrates that one cannot guarantee more than one contractible edge. However, we are not aware of any graph on more than 6 vertices that admits a depth-first search tree containing exactly one contractible edge.

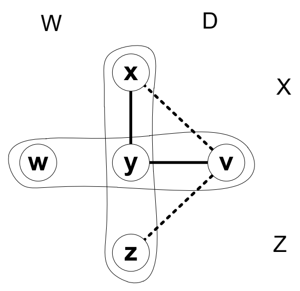


Figure 5.4: The T-minimal split component D.

Consider a 3-connected graph G on more than 4 vertices. Assume that G is a fox and let T be a spanning tree of G containing no contractible edge. It follows that, for every edge $xy \in E(T)$, there exists a vertex $z \in V(G)$ such that $\{x, y, z\}$ is a separation triple. We call $\{x, y, z\}$ a T-separation triple (this assumes $xy \in E(T)$). Split components that result from the removal of a T-separation triple are called T-split components. A *T-minimal split component* is a T-split component that does not properly contain a T-split component.

Lemma 105. *Let G be a 3-connected graph on more than 4 vertices. Assume that G is a fox and let T be a spanning tree of G containing no contractible edge. Then, every T-minimal split component consists of exactly one vertex, say v. This vertex has degree 3 and is incident to exactly one edge of T. More precisely, if the neighbors of v in G are x, y, and z with $xy \in E(T)$, then $vz \notin E(T)$ and either $vx \in E(T)$ or $vy \in E(T)$.*

Proof. Let D be a T-minimal split component and let $\{x, y, z\}$ with $xy \in E(T)$ be the associated separation triple. Since T is a spanning tree, there exists a vertex $v \in V(D)$ that is a neighbor of x, y, or z in T. We show that D has only one vertex, namely v.

If $vz \in E(T)$, then vz is non-contractible and hence a separation triple $\{v, z, w\}$ exists. Since $xy \in E(G)$, either $w \in \{x, y\}$ or both, x and y, are in the same split component of $G \setminus \{v, z, w\}$. Consequently, there exists a split component S of $G \setminus \{v, z, w\}$ such that $x, y \notin V(S)$. By Lemma 99, v has a neighbor, say u, in S. Since $u \notin \{x, y, z\}$, u is in the same split component of $G \setminus \{x, y, z\}$ as v, i. e., $u \in V(D)$. It follows that every vertex in S is in D as well. Since $v \notin V(S)$, S is properly contained in D, which is a contradiction to the minimality of D. It follows that $vz \notin E(T)$. Accordingly, either $vx \in E(T)$ or $vy \in E(T)$.

Assume w.l.o.g. that $vy \in E(T)$. See Figure 5.4. Therefore, vy is non-contractible and a separation triple $\{v, y, w\}$ exists. If there is a split component of $G \setminus \{v, y, w\}$ that contains neither x nor z, the arguments of the preceding paragraph indicate that the T-split component D is not T-minimal. It follows that $\{v, y, w\}$ splits G into exactly two components, one containing x and one containing z. Call the former component X and the latter Z. We show next that both, $X \cap D$ and $Z \cap D$, must be empty.

If $X \cap D \neq \emptyset$, Lemma 102 implies that $\{x, y, v\}$ separates $X \cap D$ from the rest of G, which contradicts the minimality of D. This implies that $X \cap D = \emptyset$. Analogously, $Z \cap D = \emptyset$.

Thus, we have shown that, assuming $v \in V(G)$ is in a T-minimal split component, there exists a separation triple $\{x, y, z\}$ with $xy \in E(T)$ such that $vx, vy, vz \in E(G)$, $\deg(v) = 3$, $vz \notin E(T)$ and w.l.o.g. $vy \in E(T)$. □

Although foxes must have some vertices of degree 3, as indicated by the previous lemma, not all vertices of a fox can be of degree 3.

Theorem 106. *If G is a 3-connected 3-regular graph on more than 4 vertices, then G is not a fox.*

Proof. Assume that G has a spanning tree T containing no contractible edge. According to Lemma 105, there are vertices $v, x, y, z \in V(G)$ such that $vx, vy, vz \in E(G)$, $xy, vy \in E(T)$ but $vz \notin E(T)$. Because G is 3-regular, $deg(x) = deg(y) = 3$. As T is a spanning tree of G, either the third edge incident to x, say xr, or the third edge incident to y, say ys, is a tree edge. Since $vy \in E(G)$, xr is contractible by Lemma 98. Since $xy, vy \in E(T)$, both edges are non-contractible by assumption. Accordingly, ys is contractible by Lemma 97. This contradicts the assumption that T contains no contractible edge. □

Consider a 3-connected graph G on more than 4 vertices. Assume that G is a fox and let T be a spanning tree of G that contains no contractible edge. Let v be a T-minimal split component in G and let vy be the only tree edge incident to v. We call a T-separation triple $\{v, y, w\}$ a *special T-separation triple*. Split components that result from deleting a special T-separation triple are called *special T-split components*. A *special T-minimal split component* is a special T-split component that does not properly contain a special T-split component.

Lemma 107. *Let G be a 3-connected graph on more than 4 vertices. Assume that G is a fox and let T be a spanning tree of G that contains no contractible edge. Then, every special T-minimal split component consists of exactly one vertex and has a neighbor that is also a special T-minimal split component. Let v and v' be such a pair of special T-minimal split components with $vv' \in E(G)$. Then, G contains a vertex y such that $vy, v'y \in E(T)$.*

Proof. Let X be a special T-minimal split component; X is split off by the special T-separation triple $ST = \{v, y, w\}$ with $vy \in E(T)$ and such that v is a T-minimal split component. According to Lemma 100, the three vertices of no other special T-separation triple are contained in $V(X) \cup ST$. Since ST is a T-separation triple, there is a T-minimal split component $v' \in V(X)$; v' belongs to a special T-separation

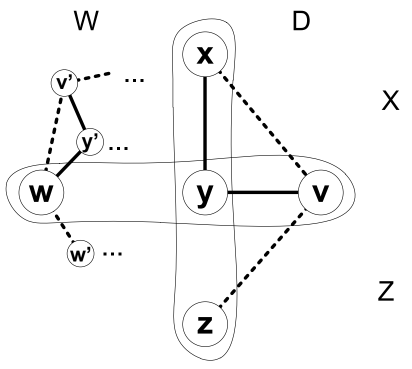


Figure 5.5: A case contradicting the minimality of X

triple $ST' = \{v', y', w'\}$ with $v'y' \in E(T)$, where $y' \in V(X) \cup ST$ and $w' \notin V(X) \cup ST$ (otherwise, X would not be minimal).

Assume first that $y' \in V(X)$. Then, w' must split $G \setminus X$, and Lemma 103 implies that one of the vertices in ST has w' as its only neighbor in $G \setminus X$. Since $vy \in E(G)$, such vertex must be w. We next show that all neighbors of w are contained in ST', implying that w has degree 3. Assume to the contrary that w has a neighbor $u' \notin ST'$. Then, u' and w belong to the same split component of $G \setminus ST'$. Every path from u' to any vertex in a different split component of $G \setminus ST'$ must pass through either v', y' or w. Hence, $\{v', y', w\}$ is a special T-separation triple contained in $V(X) \cup ST$. But such possibility is ruled out in the previous paragraph because of the minimality of X.

It follows that w has degree 3, its neighbors are precisely the vertices in ST', and w is a T-minimal split component. By Lemma 98, ww' is contractible, and accordingly does not belong to T. Additionally, $wv' \notin E(T)$, since $v'y' \in E(T)$ and v' has only one incident tree edge. Hence, $wy' \in E(T)$. Let z' be the third neighbor of v' besides y' and w. Then, $\{w, y', z'\}$ is a special T-separation triple that separates v' from the rest of G (see Figure 5.5). This contradicts our choice of X being minimal. We conclude that $y' \notin V(X)$, and hence $y' \in ST$.

Since v' and w' are in different split components of $G \setminus ST$, the triples ST and ST' cross due to Lemma 101. Hence, the vertices of $ST \setminus \{y'\}$ must belong to different split components of $G \setminus ST'$. Since $vy \in E(G)$, this excludes the possibility that $y' = w$. Moreover, $y' \neq v$, since otherwise v would be incident to two tree edges, namely vy and $v'y'$. It follows that $y = y'$. If $|V(X)| > 1$, Lemma 102 implies that either $\{v', y, v\}$ or $\{v', y, w\}$ is a special T-separation triple. Such a triple has a split component properly contained in X, which contradicts the minimality of X. It follows that v' is the only vertex in X. Let w'' be the third neighbor of v besides v' and y. Then, $\{v', y, w''\}$ is a special T-separation triple that separates v from the rest of G. We conclude that v and v' are both special T-minimal split components, $vv' \in E(G)$ and $vy, v'y \in E(T)$. □

Theorem 108. *Let G be a 3-connected graph on more than 4 vertices. Assume that G is a fox and let T be a spanning tree of G that contains no contractible edge. Then, G contains two edges whose four end vertices are distinct special T-minimal split components.*

Proof. Let v and v' be adjacent special T-minimal split components as described in Lemma 107; v is split off by $ST' = \{v', y, w'\}$ and v' is split off by $ST = \{v, y, w\}$.

Assume first that there is a special T-minimal split component in $V(G) \setminus \{v, v', y, w, w'\}$. Call it z, and let z' be the adjacent special T-minimal split component. Then, $z' \notin \{v, v'\}$, and hence (v, v') and (z, z') are the desired pairs.

Otherwise, any special T-minimal split component of G is contained in $\{v, v', y, w, w'\}$. Let W' be the component of $G \setminus ST$ that contains w' and let W be the component of $G \setminus ST'$ that contains w. Both, W' and W, are special T-split components and hence contain special T-minimal split components. These components must be w for W and w' for W'. Then, (v, w') and (v', w) are the desired pairs. □

Next, we use Theorem 108 to prove our main result.

Theorem 109. *Let G be a graph on more than 4 vertices. Every depth-first search tree of G contains a contractible edge.*

Proof. Let T be a depth-first search tree of G and assume that T contains no contractible edge. According to Theorem 108, there exist two pairs of distinct vertices of degree 3, each vertex being a T-minimal split component, such that the vertices of each pair are adjacent in G. With Lemma 105, every T-minimal split component is a vertex of degree 3 that is either the root or a leaf in T. Accordingly, there exists a pair of vertices that are leaves in T while being adjacent in G, which is a contradiction to the fact that T is a depth-first search tree. □

We remark that there are arbitrarily large foxes that contain exactly four vertices of degree 3 (see Figure 5.6).

5.5 Conclusions

The main result of this chapter is that every depth-first search tree of a 3-connected graph contains a contractible edge. However, not every spanning tree of a 3-connected graph contains a contractible edge. An interesting fact is that all wheel graphs, as well as the members of the infinite family of foxes in Figure 5.2, satisfy the equation $m = 2n - 2$. This raises the question about the existence of an infinite family of foxes where $|m - 2n|$ grows large, e. g., is not bounded by any constant. So far we have only found foxes with $|m - 2n| \leq 3$ (see Figure 5.7 for an extremal example). Another open question would be whether there exists an inductive characterization

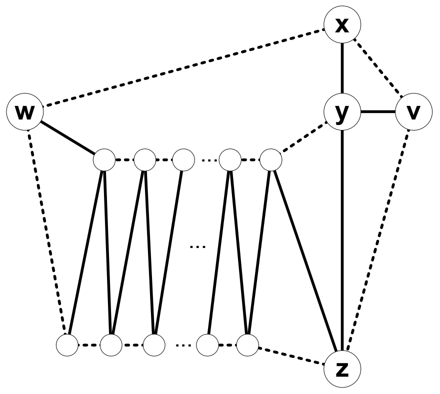


Figure 5.6: An arbitrarily large fox with exactly four degree-3 vertices.

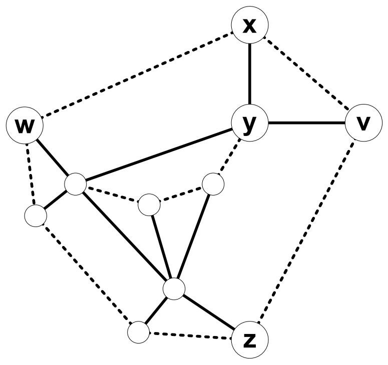


Figure 5.7: A fox with $m = 2n - 3$.

of foxes. Such a characterization would provide more insight into the distribution of contractible edges in 3-connected graphs.

Bibliography

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

[2] K. Ando, H. Enomoto, and A. Saito. Contractible edges in 3-connected graphs. *J. Comb. Theory, Ser. B*, 42(1):87–93, 1987.

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

[4] D. W. Barnette. Conjecture 5. In W. T. Tutte, editor, *Recent Progress in Combinatorics. Proceedings of the 3rd Waterloo Conference on Combinatorics*, volume 3, page 343, 1969.

[5] D. W. Barnette and B. Grünbaum. On Steinitz's theorem concerning convex 3-polytopes and on some properties of 3-connected graphs. *Many Facets of Graph Theory, Lecture Notes in Mathematics*, 110:27–40, 1969.

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

[7] 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.

[8] G. D. 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.

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

[10] 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.

[11] J. A. Bondy and U. S. R. Murty. *Graph Theory*, volume 244 of *Graduate texts in mathematics.* Springer, 2008.

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

[13] 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.

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

[15] R. Diestel. *Graph Theory.* Springer, third edition, 2005.

[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*, to appear.

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

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

[20] K. A. Frenkel. An interview with the 1986 A. M. Turing Award recipients — John E. Hopcroft and Robert E. Tarjan. *Commun. ACM*, 30(3):214–222, 1987.

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

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

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

[24] 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.

[25] 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.

[26] R. L. Graham, M. Grötschel, and L. Lovász, editors. *Handbook of Combinatorics, Volume I.* Elsevier (North-Holland); The MIT Press, 1995.

[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, London, UK, 2001. Springer-Verlag.

[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] R. Halin. A theorem on n-connected graphs. *Journal of Combinatorial Theory*, 7(2):150–154, 1969.

[30] R. Halin. Zur Theorie der n-fach zusammenhängenden Graphen. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 33(3):133–164, 1969.

[31] F. Harary. *Graph Theory.* Addison-Wesley, Reading, MA, first edition, 1969.

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

[33] D. A. Holton, B. Jackson, A. Saito, and N. C. Wormald. Removable edges in 3-connected graphs. *J. Graph Theory*, 14(4):465–473, 1990.

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

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

[36] 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.

[37] H. Kang, J. Wu, and G. Li. Removable edges of a spanning tree in 3-connected 3-regular graphs. In *Frontiers in Algorithmics, LNCS 4613*, pages 337–345, 2007.

[38] A. K. Kelmans. Graph expansion and reduction. *Algebraic methods in graph theory, Szeged, Hungary*, 1:317–343, 1978.

[39] 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. Preliminary version in SODA 2003, pp. 158–167.

[40] M. Kriesell. A survey on contractible edges in graphs of a prescribed vertex connectivity. *Graphs and Combinatorics*, 18(1):1–30, 2002.

[41] M. Kriesell. On the number of contractible triples in 3-connected graphs. *J. Comb. Theory, Ser. B*, 98(1):136–145, 2008.

[42] 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.

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

[44] R. M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying algorithms. *Computer Science Review*, 2010. to appear.

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

[46] 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.

[47] 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.

[48] 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.

[49] K. Menger. Zur allgemeinen Kurventheorie. *Fund. Math.*, 10:96–115, 1927.

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

[51] 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.

[52] 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.

[53] H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(1-6):583–596, 1992.

[54] 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.

[55] K. Ota. The number of contractible edges in 3-connected graphs. *Graphs and Combinatorics*, 4(1):333–354, 1988.

[56] J. G. Oxley. *Matroid Theory.* Oxford University Press, 1992.

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

[58] A. Schrijver. Paths and Flows — A historical survey. *CWI Quarterly*, 6(3):169–183, 1993.

[59] P. D. Seymour. Sums of circuits. In *Graph Theory and Related Topics*, pages 341–355, Univ. Waterloo, 1979.

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

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

[62] G. Szekeres. Polyhedral decompositions of cubic graphs. *Bull. Austral. Math. Soc.*, 8:367–387, 1973.

[63] 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.

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

[65] R. E. Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2(6):160–161, 1974.

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

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

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

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

[70] V. K. Titov. *A constructive description of some classes of graphs.* PhD thesis, Moscow, 1975.

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

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

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

[74] Y. H. Tsin and F. Y. Chin. A general program scheme for finding bridges. *Information Processing Letters*, 17(5):269–272, 1983.

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

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

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

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

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

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

[81] D. B. West. *Introduction to Graph Theory.* Prentice Hall, 2001.

[82] H. Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932.

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

[84] 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.

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

Index