Certifying 3-Connectivity in Linear Time

Jens M. Schmidt
MPI für Informatik, Saarbrücken, Germany

Abstract

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

We show how to compute both sequences in optimal time, improving the previously best known running times of $O(|V|^2)$ to $O(|E|)$. Based on this result, we give a linear-time test of 3-connectivity that is certifying; finding such an algorithm has been a major open problem in the design of certifying algorithms in the last years. The 3-connectivity test is conceptually different from well-known linear-time tests of 3-connectivity; it uses a certificate that is easy to verify in time $O(|E|)$. We also provide an optimal certifying test of 3-edge-connectivity.

1 Introduction

The class of 3-connected (i. e., 3-vertex-connected) graphs has been studied intensively for many reasons in the past 50 years. Besides being a fundamental graph property, 3-connectivity has numerous applications, in particular (but not only) for problems in graph drawing (see [14] for a survey), problems related to planarity and online problems on planar graphs (see [3] for a survey).

We use graph constructions throughout the paper. Let B be a set of graphs, G be a graph and O be a finite set of graph operations. A sequence

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

of operations of O that generates G when applied to a graph of B is called a *construction sequence from B to G (using O)*. We will call B the set of *base graphs*. When B and O are clear from the context, we just refer to a *construction sequence of G*. Such a sequence can also be described by giving the inverse operations from G to a base graph; we call this the *top-down* variant of a construction sequence, as opposed to a *bottom-up* variant.

The importance of construction sequences for this paper stems mainly from the fact that they certify 3-connectivity; it is however the author's belief that the inductive nature of construction sequences is a very useful, yet not fully utilized, framework to solve computational graph problems efficiently.

One of the most noted constructions for 3-connected graphs was given by Tutte [19]: Every 3-connected graph G on more than 4 vertices contains a *contractible* edge, i. e., an edge whose contraction generates a 3-connected graph. Iteratively contracting such an edge yields a top-down construction sequence from G to a K_4-multigraph. Unfortunately, also non-3-connected graphs can contain contractible edges, but adding a side condition establishes a full characterization [6]: 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 using contractions on edges with both end vertices having at least 3 neighbors; we will call this a *sequence of contractions*. In fact, the existence of the bottom-up variant of this sequence is commonly stated as Tutte's famous *wheel theorem* [19].

Barnette and Grünbaum [2] and Tutte [20] prove that every 3-connected graph G on more than 4 vertices contains a *removable* edge, i. e., an edge whose deletion generates a subdivision of a 3-connected graph. Let *removing* an edge e be the operation that deletes e and, for each end vertex v of e with exactly two distinct neighbors x and y in the remaining graph, deletes v and inserts the edge xy. Removing a removable edge leads, similar as in the sequence of contractions, to a top-down construction sequence from G to K_4; we will call this a *sequence of removals*. Again, adding a side condition fully characterizes the 3-connected graphs [18].

Although both existence theorems on contractible and removable edges are used frequently, the first non-trivial computational result to create the corresponding construction sequences was published more than 45 years afterwards: In 2006, Albroscheit [1] showed how to compute a construction sequence for 3-connected graphs in time $O(|V|^2)$. However, in this algorithm, contractions and removals are allowed to intermix. In 2010, two results [13, 18] were given that both computed a sequence of contractions in time $O(|V|^2)$. The latter result also gives an algorithm that computes a sequence of removals in $O(|V|^2)$. All mentioned algorithms do not rely on the 3-connectivity test of Hopcroft and Tarjan [9], which runs in linear time but is rather involved. No algorithm is known that computes any of these sequences in subquadratic time.

We give an algorithm that computes a sequence of removals in linear time. This will also imply a linear-time algorithm that computes a sequence of contractions (in the bottom-up and top-down variants) and has a number of consequences.

Certifying 3-connectivity in linear time. Mehlhorn and Näher [12] (see [11] for a survey) introduced the concept of *certifying algorithms*, i. e., algorithms that produce with each output a *certificate* that the particular output has not been compromised by a bug. Such a *certificate* can be any data that allows to check the correctness of the particular output (uniformly using a verifying algorithm), but should allow for an easy verification. 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 little work has been devoted to certify 3-connectivity, although sophisticated linear-time recognition algorithms are known for over 35 years [9, 17, 21]. However, none of them describes an easy-to-verify certificate.

The currently fastest algorithms that certify 3-connectivity need $\Theta(|V|^2)$ time and use construction sequences as certificates [1, 13, 18]. Recently, a linear time certifying algorithm for 3-connectivity has been proposed for the subclass of Hamiltonian graphs, when a Hamiltonian cycle is given [6]. In general, finding a certifying algorithm for 3-connectivity in subquadratic time is a major open problem [11, Chapter 5.4] [6].

We give a linear-time certifying algorithm for 3-connectivity that uses a sequence of removals as certificate. This implies a new linear-time 3-connectivity test that neither assumes the graph to be 2-connected nor needs to compute low-points (see [9] for a definition); instead, it uses the structure of 3-connected graphs implicitly by applying simple path-generating rules. This is conceptually different from all previous linear-time 3-connectivity tests. The algorithm has already been implemented and made publicly available [15]; interestingly, it outperforms the test in [9] on no-instances.

Certifying 3-edge-connectivity in linear time. There is no test for 3-edge-connectivity that is certifying and runs in linear time, although many non-certifying linear-time algorithms for this problem are known, the first being [7]. Based on a reduction in [7], we give a linear-time test on 3-edge-connectivity that is certifying.

Applications. Certifying 3-connectivity allows to make many graph algorithms that use the *SPQR-tree* data structure [8] certifying (e. g., [3, 14]). Moreover, algorithms on *polytopes* can be augmented with a quick and easy check that their input represents indeed a polytope. Applications in communication networks include certificates for their *reliability* and the property to admit a *perfectly secure message transmission* [5].

We use standard graph-theoretic terminology from [4]; let $n = |V|$ and $m = |E|$. Let $v \rightarrow_G w$ denote a path P from vertex v to vertex w in a graph G and let $s(P) = v$ and $t(P) = w$ be the *source* and *target* vertex of P, respectively (this imposes an orientation from $s(P)$ to $t(P)$ on P). Every vertex in $P \setminus \{s(P), t(P)\}$ is called an *inner vertex* of P. For $v \in V(G)$, let $N(v) = \{w \mid vw \in E\}$ (possibly $v \in N(v)$) and $deg(v)$ its degree (counting multiedges). Let $\delta(G)$ be the minimum degree in G. Let T be an undirected tree rooted at r. For two vertices x and y in T, let x be an *ancestor* of y and y be a *descendant* of x if $x \in V(r \rightarrow_T y)$. If additionally $x \neq y$, x and y are *proper* ancestors and descendants, respectively. Let $T(x)$ be the subtree of T that contains all descendants of x. Let K_2^m be the graph on 2 vertices that contains exactly m parallel edges and no self-loops.

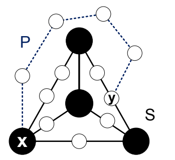
2 BG-paths

Iteratively removing removable edges in a 3-connected graph G leads to a *sequence of removals* from G to K_4, in which all generated intermediate graphs are 3-connected. However, the intermediate graphs are not necessarily subgraphs of G, which makes a linear-time computation difficult. For that reason, we reduce the computation to a closely related construction sequence [18], which is described next.

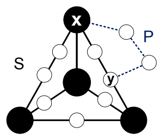
A *subdivision* of a graph G is a graph generated from G by replacing each edge of G by a path of length at least one. Let S be a subdivision of either K_2^3 or of a 3-connected graph. Let a vertex v in S be *real* if $deg(v) \geq 3$ and let $V_{real}(S)$ be the set of real vertices in S. Let the *links* of S be the paths in S that have real end vertices but contain no other real vertices. Note that the links of S are in one-to-one correspondence to the edges of the subdivided graph (which is K_2^3 or 3-connected) and thus partition $E(S)$. Let two links be *parallel* if they share the same end vertices.

Definition 1. A *BG-path* for S (see Figure 1) is a path $P = x \rightarrow_G y$, $x \neq y$, such that

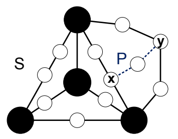
1. $V(P) \cap V(S) = \{x, y\}$
2. Every link of S that contains both x and y contains them as end vertices.
3. If x and y are inner vertices of distinct links L_x and L_y of S, respectively, and $|V_{real}(S)| \geq 4$, then L_x and L_y are not parallel.



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



(b) P is not a BG-path for S, since there is a link containing x and y such that y is no end vertex of that link (see Property 1.2).



(c) P is not a BG-path for S, as x and y are inner vertices of two links, respectively, that are parallel and $|V_{real}(S)| \geq 4$ (see Property 1.3).

Figure 1: BG-paths

It was shown in [18] that a graph G without self-loops is 3-connected if and only if $\delta(G) \geq 3$ and G can be constructed from an (arbitrary) K_4-subdivision in G by adding BG-paths. This implies that every 3-connected graph G contains a subdivision of K_4, a result first shown by J. Isbell [2]. For technical reasons, we will use a slightly modified construction that starts with a K_2^3-subdivision and demand that the first BG-path generates a K_4-subdivision. Thus, a *construction sequence using BG-paths* starts with a K_2^3-subdivision of G, adds one BG-path that generates a K_4-subdivision and then adds BG-paths until G is constructed.

Let $S_3, S_4, S_5, \ldots, S_z = G$ be the intermediate graphs that are generated by such a construction. We benefit from two key features: Each S_l, $3 \leq l < z$, is a subdivision of a 3-connected graph and, additionally, a subgraph of S_{l+1} and therefore of G. This does not only yield an easy representation in linear space, it will also allow to compute a next BG-path efficiently by searching the neighborhood of the current subgraph in G. We give old and new results about construction sequences.

Theorem 2. *The following statements are equivalent.*

(1) *A simple graph G is 3-connected*

(2) *There is a sequence of removals from G to K_4 of* removable *edges $e = xy$ with $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$ (see [18])*

(3) *There is a sequence of removals from G to K_4 of edges $e = xy$ with $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$*
(4) *There is a sequence of contractions from G to a K_4-multigraph of* contractible *edges $e = xy$ with $|N(x)| \geq 3$ and $|N(y)| \geq 3$ (see [18],* chapter 5 *in [19])*
(5) *There is a sequence of contractions from G to a K_4-multigraph of edges $e = xy$ with $|N(x)| \geq 3$ and $|N(y)| \geq 3$ (see [6])*
(6) *$\delta(G) \geq 3$ and there is a sequence of BG-paths from a K_2^3-subdivision in G to G such that the first BG-path generates a K_4-subdivision*
(7) *$\delta(G) \geq 3$ and there is a sequence of BG-paths from* a *K_4-subdivision in G to G [2, 18]*
(8) *$\delta(G) \geq 3$ and there is a sequence of BG-paths from* each *K_4-subdivision in G to G [18]*

Lemma 3. *There are algorithms that transform a sequence of Type (6) to the sequences of each of the Types (2)–(7) in linear time.*

With Lemma 3, we can transform a sequence of Type (6) to every sequence of Theorem 2 efficiently. We will therefore focus on computing this sequence; if not stated otherwise, a *construction sequence* will refer to a sequence of Type (6). The following lemma provides an iterative algorithmic approach to compute it.

Lemma 4 ([18])**.** *Let G be a 3-connected graph and $H \subset G$ such that H is a subdivision of either K_2^3 or of a 3-connected graph. There is a BG-path for H in G.*

Clearly, every sequence of Type (4) and (5) must contain exactly $n - 4$ contractions. We give a corresponding result for the number of operations in the other sequences.

Lemma 5. *Every sequence of Type (2), (3) and (7) contains exactly $m - n - 2$ operations and every sequence of Type (6) contains exactly $m - n - 1$ operations.*

3 Chain Decompositions and Certificates

We first describe a decomposition of a simple graph G, which is closely related to *ear* and *open ear* (i. e., no ear is a cycle) decompositions [10]. This decomposition will be the base structure that allows to compute a sequence of Type (6) efficiently. We define the structure algorithmically on a depth-first search (DFS) forest [9]; a similar procedure for the computation of so-called low-points (see [9]) can be found in [17]. Let F be a (rooted) DFS-forest of G. For every backedge e, let $s(e)$ and $t(e)$ denote the two end vertices of e such that $s(e)$ is a proper ancestor of $t(e)$ in F.

We decompose G into a set $C = \{C_1, \ldots, C_{|C|}\}$ of cycles and paths, called *chains*, by applying the following procedure for each vertex v in DFS-order (see also Figure 2): Let T be the tree in F that contains v and let r be the root of T. For every backedge vw with $s(vw) = v$, we traverse the path $w \to_T r$ until a vertex x is found that is either r or already contained in a chain. The traversed subgraph $vw \cup (w \to_T x)$ forms a new *chain* C_i with $s(C_i) = v$ and $t(C_i) = x$. We call C a *chain decomposition*. Let $<$ be the strict total order on C in which the chains were found, i. e., $C_1 < \cdots < C_{|C|}$. Clearly, the decomposition into chains can be computed in time $O(n + m)$.

Interestingly, C is an open ear decomposition if and only if G is 2-connected, an ear decomposition if and only if G is 2-edge-connected and we can test both facts by checking very easy conditions on C in linear time (proofs omitted). Thus, chain decompositions unify existing linear-time tests on 2-connectivity and 2-edge-connectivity without the necessity to compute low-points in advance. If G is not 2-(edge)-connected, a cut vertex (respectively, a bridge) can be found in linear time.

Easy-to-verify certificates for low (edge-)connectivity. Suppose there is a vertex or edge cut X of size $k - 1 \geq 0$ 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-(edge-)connected. However, certificates should be as easy to check as possible, while the running time for computing them is less important. We thus apply a 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 (we call this a *red-green coloring*). A checker for G being not k-connected then just needs to check that at most $k - 1$ vertices are uncolored, there is at least one red and one green vertex and that no edge joins a red vertex with a green one. For G being not k-edge-connected, it suffices to check that there is a red, a green and no uncolored vertex and that the end vertices of at most $k - 1$ edges differ in color. The certificates need space $O(n)$ and can be checked in time $O(m)$, as $n > m + 1$ proves G to be disconnected.

A certificate for G being connected is given in [11], using an easy numbering scheme on the vertices. Easy-to-verify certificates for 2- and 2-edge-connectivity are open ear decompositions and ear decompositions [10], respectively, as computed by the chain decomposition. For 3-connectivity, we will use a sequence of Type (6) as certificate, which proves G to be 3-connected due to Theorem 2. A simple checker in $O(m)$ time for this sequence is given in [18].

For certifying 3-edge-connectivity, we use a reduction to 3-connectivity due to Galil and Italiano [7]. 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', let $v_1, \ldots, v_{deg(w)}$ be the arc-vertices neighboring w. Then the edges $(v_1v_2, v_2v_3, \ldots, v_{deg(w)}v_1)$ are added to G' if not already existent. The graph G is 3-edge-connected if and only if G' is 3-connected [7]. Moreover, every vertex cut of minimal size in G' contains only arc-vertices (Lemma 2.2 in [7]).

We now apply a certifying 3-connectivity test to G'. 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. We can then use a red-green coloring of the connected components of $G \setminus X$ as certificate.

Otherwise, G' is 3-connected and we have a sequence of Type (6) for G'. The certificate consists of this sequence, G' and the injective mapping ϕ from each vertex in G' to its corresponding vertex or edge in G to certify the construction of G'. For a checker, it suffices to test that G' is 3-connected using the given sequence, every vertex in G has a unique preimage in $V(G')$ under ϕ, every non-arc-vertex v in G' is the hub of a wheel graph with $v+1$ vertices that are all arc-vertices except for v, every two wheels in G' share at most one arc-vertex and every arc-vertex u in G' is incident to exactly two non-arc-vertices v and w such that $\phi(u) = \phi(v)\phi(w)$ and $\phi(u) \in E(G)$. Note that this checker may fail in detecting additional edges (but not additional vertices) in G and that this does not harm the 3-edge-connectivity of G. In both cases, the given certificate needs linear space and can be checked in time $O(m)$.

4 A Certifying Algorithm for 3-Connectivity in Linear Time

Due to space constraints, we give only a high level description of the certifying algorithm. According to Lemma 4, it suffices to add iteratively BG-paths to an arbitrary K_2^3-subdivision S_3 in G to get a sequence of Type (6) from S_3

to $S_z = G$. With Lemma 5, $z = m-n+2$. Note that we cannot make wrong decisions when choosing a BG-path (except for the first one), as Lemma 4 ensures a completion of the sequence if G is 3-connected. We aim for adding chains as BG-paths, as they can be efficiently computed.

We compute a chain decomposition on a DFS-forest T of G and check $n \geq 4$, $\delta(G) \geq 3$ and 2-connectivity of G as part of this computation. If the test fails, we can easily find a certificate for G being not 3-connected; otherwise, we obtain the K_2^3-subdivision $S_3 = C_1 \cup C_2$. To keep further explanations as simple as possible, we abuse notation and split the cycle C_1 into two paths by setting $C_0 = t(C_2) \rightarrow_T r$ and redefining $C_1 = r \rightarrow_{C_1 \setminus E(C_0)} t(C_2)$. We will represent the chain decomposition C as $C_0, \ldots, C_{m-n+1}$.

For every chain $C_i \neq C_0$, C_i contains exactly one backedge, namely its first edge, and $s(C_i)$ is a proper ancestor of $t(C_i)$. We define the following necessity for the 3-connectivity of G, which can be checked in linear time, giving a separation pair if violated. Recall that $T(x)$ is the subtree of T that contains all descendants of x.

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

Until now we only checked necessary properties for the 3-connectivity of G, which we will take for granted for the rest of the paper. 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. Chains admit the following tree structure.

Lemma 6. *The parent relation on C defines a tree U with $V(U) = C$ and root C_0.*

We assign one of the Types 1, 2a, 2b, 3a and 3b to each chain $C_i \neq C_0$ in ascending order of $<$. Some types will be BG-paths and therefore lead to the next subgraph in the construction sequence. The remaining ones will be grouped into bigger structures that can be decomposed into BG-paths later. Algorithm 1 defines the types in linear time; all chains are unmarked at the beginning. We illustrate the different types in Figure 2.

Algorithm 1 marks every chain of Type 2b. We explain how the algorithm groups chains of certain

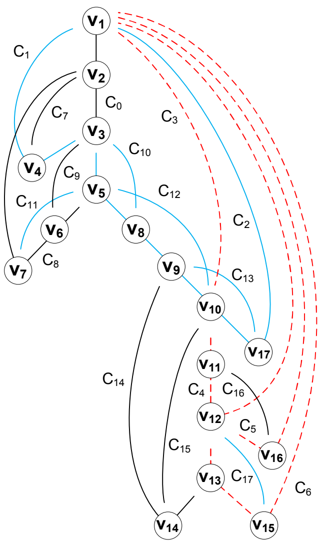


Figure 2: A chain decomposition. Light solid chains are of *Type 1*, red dashed ones of *Type 2* (*Type 2a*: C_3) and black solid ones of *Type 3* (*Type 3b*: C_{14} and C_{16}, giving the caterpillars $L_{14} = \{C_{14}, C_6, C_4\}$ and $L_{16} = \{C_{16}, C_5\}$).

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

```
1:  Let $C_k$ be the parent of $C_i$ in $U$                                    ▷ $C_k < C_i$
2:  if $t(C_i) \to_T s(C_i)$ 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 $L_i$; $C_j := parent(C_j)$
```

types. Whenever a chain C_i of Type 3b is found,
the path $C_i \to_U C_0$ is traversed until a chain C_j occurs 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). This way, every chain C_i of Type 3b is associated with a list L_i of chains; we call L_i a *caterpillar*. Property B ensures that caterpillars consist of exactly the chains in C that are of Type 2b and 3b.

In order to decide which chain can be added as BG-path, we want to impose the following structure on every graph S_l, $3 \leq l \leq m - n + 2$.

Definition 7. Let S_l be *upwards-closed* if, for each vertex v in S_l, the edge from v to its parent in T is contained in S_l. Let S_l be *modular* if S_l is the union of chains.

Clearly, S_3 is upwards-closed and modular. We would be done if we could restrict every S_l to be upwards-closed and modular, as then every BG-path would be a chain:

Lemma 8. *If S_l and S_{l+1} are upwards-closed and modular, the BG-path P for S_l is a chain.*

Proof. Assume that P is not a chain. Since S_{l+1} is modular, P must be the union of $t > 1$ chains forming a path; let C_i be the first chain in P. Now P cannot start with $t(C_i)$, as Property 1.1 and $s(C_i) \in V(S_l)$ would force C_i to be the only chain in P, contradicting $t > 1$. Thus, P starts with $s(C_i)$ and, for the same reason, $(t(C_i) \to_T s(C_i)) \nsubseteq S_l$. Since S_{l+1} is upwards-closed, $(t(C_i) \to_T s(C_i)) \subseteq S_{l+1}$. This contradicts $t > 1$ as well, as a chain in P that contains an edge of $t(C_i) \to_T s(C_i)$ would induce a vertex of degree at least 3 in the path P, because it contains a backedge. □

Unfortunately, restricting every S_l to be upwards-closed and modular is not possible, as the 3-connected graph in Figure 3 shows: Since every BG-path for S_3 has end vertices x and y, S_4 cannot be modular. We therefore aim to restrict only certain subgraphs. Let a *cluster* be either a caterpillar or a chain of Type 1, 2a or 3a (the *cluster of a chain* is the cluster containing the chain). Instead of adding BG-paths one by one, we will add clusters that can be decomposed into subsequent BG-paths later; we restrict only the subgraph obtained from the last BG-path to be upwards-closed and modular. We list the restrictions for adding a cluster in detail.

Restrictions: We add a cluster to an upwards-closed modular subgraph S_l only if it

(R_1) can be decomposed into as many subsequent BG-paths as it contains chains and creates an upwards-closed and modular subgraph S_{l+t}, $t > 0$, 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 $S_{l+t} \neq S_3$).

Finding such a cluster clearly gives the next t BG-path(s) for S_l. In particular, (R_1) ensures that the total number of BG-operations is $|C| - 3 = m - n - 1$, as shown in Lemma 5. Restriction (R_2) implies that the first BG-path generates a K_4-subdivision, as demanded for a construction sequence, and not, e. g., a K_2^4-subdivision. We will assume from now on that S_l was obtained obeying Restrictions (R_1) and (R_2). Let a cluster for S_l that satisfies (R_1) and (R_2) be *addable*. In the following, we investigate how a set of addable clusters for S_l can be obtained.

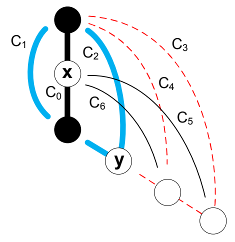


Figure 3: No BG-path for S_3 (thick subgraph) preserves modularity.

Definition 9. For S_l and a chain C_i in S_l, 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 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.

We process chains in the order $<$ of creation, i. e., top-down in the tree U. The key idea for each C_i is to add (among others) the clusters of all chains in $\mathit{Children}_{12}(C_i)$ and the clusters of all chains in $\mathit{Type}_3(C_i)$. Note that we defined $\mathit{Children}_{12}(C_i)$ to contain only chains of Type 1 and 2. It will not be necessary to consider children of C_i of Type 3, as their clusters will be added before as part of $\mathit{Type}_3(C_j)$ for an ancestor C_j of C_i in U. The sets $\mathit{Children}_{12}(C_i)$ and $\mathit{Type}_3(C_i)$ can be computed efficiently. We give the precise set of clusters we add.

Definition 10. For S_l and a subset $A \subseteq C$ of chains, let $cl(A)$ be the set of clusters that contains all (not necessarily proper) ancestors of chains in A that are not contained in S_l. For every C_i, we add the clusters in $cl(\mathit{Children}_{12}(C_i) \cup \mathit{Type}_3(C_i))$.

Theorem 11. *Let C_i be a chain in S_l with* $\text{Children}_{12}(C_j) = \text{Type}_3(C_j) = \emptyset$ *for every proper ancestor C_j of C_i. If G is 3-connected, there is an order in which the clusters in* $\text{cl}(\text{Children}_{12}(C_i) \cup \text{Type}_3(C_i))$ *are successively addable.*

Assume for a moment that G is 3-connected. Clearly, C_0 satisfies the precondition of Theorem 11 for S_3. By induction, let the precondition be true for every C_j, $j \leq i$. Applying Theorem 11 on C_i then generates a subgraph, in which the precondition is satisfied for C_{i+1}. This ensures that iteratively applying Theorem 11 on $C_0, C_1, \ldots, C_{m-n+1}$ constructs G. We obtain the following corollary.

Corollary 12. For every 3-connected graph G there is a sequence of Type (6) to G that is restricted by (R_1) and (R_2).

4.1 Reduction to Overlapping Intervals

Theorem 11 provides an algorithmic method to compute a construction sequence: For each C_i, $0 \leq i \leq m-n+1$, we add the clusters in $cl(\mathit{Children}_{12}(C_i) \cup \mathit{Type}_3(C_i))$; we say that C_i is *processed*. We describe the processing phase of C_i (see Algorithm 2). Let S_l be the current subgraph. Theorem 11 does not state in which order the clusters are addable; it therefore remains to show how we can compute this order if exists. We first partition the chains in $\mathit{Type}_3(C_i)$ into so-called segments of S_l.

Definition 13. Let E' be a maximal subset of $E(G) \setminus E(S_l)$ such that every two edges of E' are contained in a path whose inner vertices are disjoint from $V(S_l)$. Then the edge-induced subgraph $G[E']$ is called a *segment* of S_l. Let the *segment of a chain* $C_i \nsubseteq S_l$ be the segment of S_l that contains C_i.

Note that every segment of S_l is the union of all vertices in a subtree of U, as S_l is upwards-closed and modular. A segment can therefore be represented by the minimal chain it contains. Let X be the subset of chains

Algorithm 2 Certify3Connectivity(Graph G)

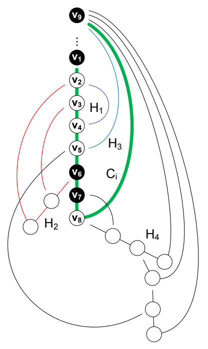
```
1: Compute a DFS-tree $T$ of $G$, a chain decomposition $C$, classify the chains and check simple properties
2: Check Property B, Set $S_3 := C_0 \cup C_1 \cup C_2$    ▷ Page 9 and Section 3
3: for $i := 0$ to $m - n + 1$ do    ▷ process each $C_i$ and add clusters of Theorem 11
4:     Compute the lists $\mathit{Children}_{12}(C_i)$ and $\mathit{Type}_3(C_i)$
5:     Partition $\mathit{Type}_3(C_i)$ into segments
6:     $X :=$ subset of chains in $\mathit{Type}_3(C_i)$ whose segments do not contain a chain of $\mathit{Children}_{12}(C_i)$
7:     Add the clusters in $\mathit{cl}(X)$ successively in the order of $<$; update $\mathit{Type}_3(C_i)$
8:     $Y :=$ set of segments that contain a chain in $\mathit{Children}_{12}(C_i)$
9:     for each segment $H \in Y$ do
10:        Compute the attachment vertices of $H$ and the dependent path of $H$
11:        Map $H$ to a set of intervals on $C_i$    ▷ Section 4.1
12:    if the merged overlap graph $G'$ of $Y$ is connected then
13:        Obtain a proper order $\sigma$ on $Y$ from $G'$    ▷ Lemma 15
14:        for each segment $H \in Y$ in the order of $\sigma$ do    ▷ Add clusters; save construction seq.
15:            Add the clusters in $\mathit{cl}(\mathit{Type}_3(C_i) \cup \mathit{Children}_{12}(C_i))$ that are in $H$ in the order of $<$
16:    else
17:        Compute a separation pair    ▷ $G'$ is not 3-connected
```

in $\mathit{Type}_3(C_i)$ whose segments do not contain a chain in $\mathit{Children}_{12}(C_i)$. Then the clusters in $\mathit{cl}(X)$ are successively addable in the order of $<$. We just add them in this order and delete X from $\mathit{Type}_3(C_i)$. For convenience, we abuse notation and let S_l be again the current subgraph.

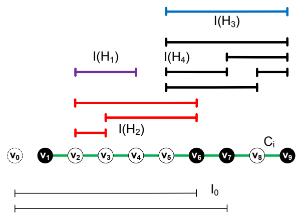
Let Y be the set of segments that contain a chain in $\mathit{Children}_{12}(C_i)$. Note that every cluster that we still have to add in this processing phase is contained in one segment in Y. For each segment $H \in Y$, let $H \cap S_l$ be the *attachment vertices* of H. It can be deduced from H containing a chain in $\mathit{Children}_{12}(C_i)$ that all attachment vertices of H are contained in C_i (see, e. g., Figure 4(a)). Let the maximal path in C_i that connects two attachment vertices of H be the *dependent path* of H. For example, the dependent path of H_4 in Figure 4(a) is $v_5 \rightarrow_{C_i} v_9$.

Consider a segment $H \in Y$ and its dependent path P. It can be shown that the clusters in $\mathit{cl}(\mathit{Children}_{12}(C_i) \cup \mathit{Type}_3(C_i))$ that are contained in H are successively addable in the order of $<$ if P contains an inner real vertex. Moreover, if P does not contain an inner real vertex, none of these clusters is addable. Whenever we have found a segment with an inner real vertex in its dependent path, we will therefore add all clusters in this segment successively.

Note that adding the clusters of a segment H causes all attachment vertices of H to be real. This might induce new inner real vertices for dependent paths of other segments in Y. It remains to compute in which order the segments of Y can be added such that every dependent path will have an inner real vertex if possible. Let an order σ on Y be *proper* if the dependent path of each segment in σ contains an inner real vertex



(a) C_i and the partition of the clusters in $cl(\mathit{Type}_3(C_i) \cup \mathit{Children}_{12}(C_i))$ into segments.



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

Figure 4: Mapping the segments H_{1-4} to intervals on C_i. Different colors depict different segments.

or an inner vertex that is an attachment vertex of a previous segment in σ. A proper order on Y thus gives the desired order on all clusters in $cl(\mathit{Children}_{12}(C_i) \cup \mathit{Type}_3(C_i))$.

We describe how to compute a proper order σ efficiently, if exists. This is the heart of the reduction. We map each segment H in Y to a set $I(H)$ of intervals on $V(C_i)$: Let $a_1, \ldots, a_k$ be the attachment vertices of H and let $I(H) = \bigcup_{1<j\le k}\{[a_1, a_j]\} \cup \bigcup_{1<j<k}\{[a_j, a_k]\}$ (see Figure 4). Additionally, we augment C_i by an artificial vertex v_0 (next to $t(C_i)$) and map the real vertices $b_1, \ldots, b_k$ of C_i to the set of intervals $I_0 = \bigcup_{1<j<k}\{[v_0, b_j]\}$. The intervals can be efficiently computed; there are at most $|\mathit{Children}_{12}(C_i)| + 2|\mathit{Type}_3(C_i)| + |V_{real}(C_i)| - 2$ intervals for C_i, giving a total of $O(m)$ intervals for all processing phases.

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 that starts with an interval in I_0. Let the *overlap graph* of Y be the graph with vertex set $I_0 \cup \bigcup_{H \in Y} I(H)$ and an edge between two vertices if and only if the corresponding intervals overlap. Let the *merged overlap graph* of Y be the graph that results from the overlap graph by merging the vertices in I_0 and in $I(H)$, respectively, to one vertex, for every segment $H \in Y$.

Lemma 14. *There is a proper order on the segments in Y if and only if the merged overlap graph G' of Y is connected.*

Clearly, the overlap graph (and the merged overlap graph) can have a quadratic number of edges in the number of intervals, e. g., consider k pairwise distinct intervals of the same length lying very close to each other. Interestingly, the connected components of the merged overlap graph can still be computed in linear time, without the need to construct the graph itself. The key idea is to use a modified variant of a sweep-line algorithm in [16] that computes the connected components of interval overlap graphs by selecting only sparse subgraphs for each component. If there is only one component, a proper order on Y can be obtained from the sparse subgraph of that component.

Lemma 15. *Let k be the number of intervals that have been created for the segments in Y and let G' be the merged overlap graph of Y. There is an algorithm with running time $O(k + |V(C_i)|)$ that computes a proper order σ on Y, if it exists, and that computes the connected components of G', if no proper order on Y exists.*

This computes the desired order. With Lemma 3, we obtain the following theorem.

Theorem 16. *The sequence of each of the types (2)–(7) for a simple 3-connected graph G can be computed in time $O(m)$.*

It is possible to extend the algorithm to certify non-3-connectivity: If the algorithm of Lemma 15 outputs more than one connected component of the merged overlap graph, a separation pair can be computed.

Theorem 17. *There are certifying algorithms for testing the 3-connectivity and 3-edge-connectivity of graphs G in time $O(n + m)$ using the certificates of Section 3.*

References

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

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

[3] 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.

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

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

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

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

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

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

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

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

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

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