An $O(n+m)$ Certifying Triconnnectivity Algorithm for Hamiltonian Graphs

Amr Elmasry* Kurt Mehlhorn† Jens M. Schmidt‡

August 26, 2010

Abstract

A graph is triconnected if it is connected, has at least 4 vertices and the removal of any two vertices does not disconnect the graph. We give a certifying algorithm deciding triconnectivity of Hamiltonian graphs with linear running time. If the input graph is triconnected, the algorithm constructs an easily checkable proof for this fact. If the input graph is not triconnected, the algorithm returns a separation pair.

1 Introduction

Hopcroft and Tarjan [HT74] and Miller and Ramachandran [MR92] gave linear time algorithms for deciding triconnectivity of graphs. If the input graph is not triconnected, both algorithms output a separation pair, i.e., a pair of vertices whose removal splits the graph into two or more components and hence witnesses that the graph is not triconnected. If the input graph is triconnected, both algorithms simply state that the graph is triconnected. Of course, the question arises whether this answer can be trusted. Indeed, Gutwenger and Mutzel [GM00] implemented the former algorithm and report that some non-triconnected graphs are declared triconnected by the algorithm. They provide a correction.

The concept of *certifying algorithms* aims at avoiding this situation. A certifying algorithm returns in addition to the required output (here: is the input graph triconnected or not?) a proof for its answer; see [MMNS10] for a general discussion of certifying algorithms. We aim for a linear-time certifying algorithm for triconnectivity. In this paper we provide such an algorithm for Hamiltonian graphs.

Tutte [Tut61] proved the existence of a linear size witness for triconnectivity: Any triconnected graph $G \neq K_4$ can be reduced to a K_4 by a sequence of edge contractions such that no intermediate graph has a vertex of degree less than 3. We call such a sequence a *Tutte contraction sequence*. It proves triconnectivity of the input graph (see

*MPI für Informatik, Supported by an Alexander von Humboldt Fellowship.

†MPI für Informatik, Campus E1 4, 66123 Saarbrücken

‡Dept. of Computer Science, FU Berlin. This research was supported by the Deutsche Forschungsgemeinschaft within the research training group "Methods for Discrete Structures" (GRK 1408).

Theorem 3) and can be computed in $O(n^2)$ time, see [Sch10]. It is open whether it can be found in sub-quadratic time.

In this paper we give an $O(n+m)$ algorithm for a special case: the graph is Hamiltonian and a Hamiltonian cycle is provided as part of the input. In other words, the graph is a cycle plus chords (= edges connecting non-adjacent vertices of the cycle). Why is a certifying triconnectivity algorithm for Hamiltonian graphs interesting? The algorithm by Hopcroft and Tarjan is recursive. The merge step essentially amounts to checking triconnectivity of Hamiltonian graphs. The algorithm by Miller and Ramachandran is based on open-ear decomposition. If all ears are edges, the graph is Hamiltonian. We hope that the linear-time algorithm for Hamiltonian graphs is a first step towards a general sub-quadratic algorithm.

This paper is organized as follows: After some preliminaries in Section 2 we present our main result in Section 3: a linear time algorithm for finding a Tutte contraction sequence of a triconnected Hamiltonian graph. In Section 4 we show how to find separation pairs in non-triconnected Hamiltonian graphs, and in Section 5 we discuss an extension to extended Hamiltonian graphs; for this graph class we achieve a time bound of $O(n \log n + m)$. In Section 6 we give a linear time algorithm for checking whether a sequence of edges is a Tutte contraction sequence. This verification algorithm works for all graphs. Finally, Section 7 surveys results for higher connectivity.

2 Preliminaries

Let $G = (V, E)$ be a finite, undirected graph without self-loops and parallel edges. We use $n := |V|$ and $m := |E|$ to denote the number of vertices and edges of G, respectively. We denote an edge between vertices u and v by uv or (u, v). A graph is *connected* if there is a path between every two of its vertices and *disconnected* otherwise. For $k > 1$, a connected graph G is *k-connected* if $n > k$ and deleting any $k - 1$ vertices leaves a connected graph. A set of vertices whose removal disconnects the graph is called a *vertex cut*. Vertex cuts of size one, two, and three are called *separation vertices*, *separation pairs*, and *separation triples*, respectively.

Let xy be an edge of G. The contraction of xy generates a graph $G' = G/xy$ with vertex set $V(G') = V(G) \setminus \{x, y\} \cup \{v_{xy}\}$, where v_{xy} is a new vertex. The edge xy is removed and in each edge having exactly one endpoint in $\{x, y\}$, this endpoint is replaced by v_{xy}. Finally, only one edge for each set of parallel edges is kept. An edge xy is *contractible* if G/xy is triconnected. If an edge xy of a triconnected graph with $n > 4$ is not contractible, there is a separation triple containing x and y.

Theorem 1 ([Tut61]) *Every triconnected graph with $n > 4$ contains a contractible edge.*

Hence there is a sequence of edge contractions that reduces G to K_4 and going only through triconnected graphs. The latter condition is hard to check. It can be replaced by a simpler condition.

Lemma 2 *Let G be a graph with minimum degree three and let xy be an edge of G. If G is not triconnected, G/xy is not triconnected.*

Proof: Consider any separation pair $\{a, b\}$. Any component of $G \setminus \{a, b\}$ has at least two vertices since the minimal vertex degree is at least three. Thus contracting xy cannot yield a triconnected graph. □

Theorem 3 *Let $G_0 = G, G_1, \ldots, G_{n-4}$ be a sequence of graphs such that G_i is obtained from G_{i-1} by the contraction of an edge, $G_{n-4} = K_4$, and every G_i has minimum degree 3 or more. Then G is triconnected.*

Proof: K_4 is triconnected. By the preceding Lemma, there can be no i such that G_i is triconnected, but G_{i-1} is not. Thus G_0 is triconnected. □

We can slightly strengthen this theorem by restricting only the degree of the endpoints of the edge to be contracted.

Theorem 4 *Let $G_0 = G, G_1, \ldots, G_{n-4}$ be a sequence of graphs such that $G_{n-4} = K_4$ and G_i is obtained from G_{i-1} by the contraction of an edge $e = xy$ with x and y having both degree at least 3. Then G is triconnected.*

Proof: We show that every G_i has minimum degree three and then appeal to the preceding theorem. Assume otherwise. Then there must be a j, $0 \leq j \leq n-5$, such that G_{j+1} has minimum degree three but G_j has not. Let v be a vertex of degree less than three in G_j and e be the edge in G_j such that $G_{j+1} = G_j/e$. Then e is not incident to v, as both end vertices of e have degree at least three, implying that its contraction does not increase the degree of v, which contradicts the assumption. □

We call a sequence as in the two preceding theorems a *Tutte contraction sequence*. In every triconnected graph at least one edge is contractible ([Tut61]). In fact, a linear number of edges is contractible ([AES87]). We use that every depth-first-search tree contains at least one contractible edge.

Theorem 5 ([EMS10]) *Let G be a triconnected graph and let T be a DFS-tree of G. Then at least one edge of T is contractible.*

We will give a self-contained proof of this theorem for the special case of Hamiltonian graphs below, see Lemma 6.

3 A Linear-Time Certifying Algorithm for Hamiltonian Graphs

We give a linear-time certifying triconnectivity algorithm for Hamiltonian graphs. We assume that a Hamiltonian cycle is given and use a Hamiltonian path contained in this cycle as DFS-tree. More precisely, we have vertices 1 to n, tree edges $(i, i+1)$ for

$1 \leq i < n$ and a set of back edges including $(n, 1)$. A *back edge* is an edge uv with $u > v+1$; u is the source of the edge and v is the target of the edge. We use $\mathcal{B}$ to denote the set of back edges. For triconnected inputs we will construct a Tutte contraction sequence in linear time. For non-triconnected inputs we will return a separation pair. We say that a vertex is *higher* than another vertex or *above* another vertex if its number is larger.

If the input is triconnected, at least one tree edge is contractible by Theorem 5. We next give a self-contained proof of this fact for the case of Hamiltonian graphs.

Lemma 6 *Let G be a triconnected Hamiltonian graph with $n > 4$. Then at least one of the edges $(i, i+1)$, $1 \leq i < n$, is contractible.*

Proof: Assume otherwise. Then for every tree edge xy, there is a z such that $\{x, y, z\}$ is a separation triple. Let

$$D_{x,y,z} = \begin{cases} \{z+1, \ldots, x-1\} & \text{if } z < x \\ \{y+1, \ldots, z-1\} & \text{if } z > y. \end{cases}$$

We choose a tree edge xy and a vertex z such that $D := D_{x,y,z}$ has minimal cardinality. Let v be the vertex in D such that vz is a tree edge. Since vz is non-contractible, there must be a w such that $\{v, z, w\}$ is a separation triple. Since x and y are neighbors, there is a separation class, call it C, with respect to $\{v, z, w\}$ containing neither x nor y. C contains a neighbor of v (otherwise $\{z, w\}$ would be a separation pair) and hence any vertex in C is reachable from v by a path avoiding x, y, and z. Because v is contained in D, $C \subseteq D$. The containment is proper since $v \in D \setminus C$.

We may assume $z > y$. Then $v = z - 1$ and $C \subseteq D = \{y+1, \ldots, v\}$. Thus $y \leq w < v - 1$ and hence $C = D_{z,v,w}$, which contradicts the minimal cardinality of D. □

The following Lemma tells us that only edges in the vicinity of a contracted edge can become contractible after a contraction. Contracting an edge xy removes the vertices x and y and introduces a new vertex v_{xy}. We define a function S on the vertices of G that returns for any vertex z of G its representative in G/xy, namely $S(x) := S(y) := v_{xy}$ and $S(z) := z$ for $z \notin \{x, y\}$.

Lemma 7 *Let G be a triconnected Hamiltonian graph, let $(i, i+1)$ be a contractible tree edge, and let $(j, j+1)$ be a non-contractible tree edge. If $(S(j), S(j+1))$ is contractible after the contraction of $(i, i+1)$ then $j \in \{i-2, i-1, i+1, i+2\}$.*

Proof: Since $(j, j+1)$ is not contractible in G, there is a separation triple containing j and $j+1$, say $ST = \{j, j+1, z\}$. Since $(i, i+1)$ is contractible, i and $i+1$ cannot both belong to ST and hence $S(ST) := \{S(j), S(j+1), S(z)\}$ has cardinality three after the contraction. Let C_1 and C_2 be the separation classes of G with respect to ST. If the sets

$$\{S(a) \mid a \in C_i\} \setminus S(ST), \ i = 1, 2,$$

are both non-empty after the contraction, $S(ST)$ is a separation triple after the contraction. So one must become empty by the contraction and hence consist of a single vertex before the contraction. Thus either i or $i+1$ must belong to $\{j-1, j, j+1, j+2\}$. □

Assume for the moment that we have a way to test whether a tree edge is contractible. We maintain a label for each tree edge. Edges labelled "non-contractible" are non-contractible and edges labelled "unknown status" are either contractible or non-contractible. Initially, all tree edges are labelled "unknown status". As long as there are more than four vertices, we do the following. We select a tree-edge labelled "unknown status" (if the graph is triconnected, there must be such an edge by Theorem 5) and test it for contractability. If it is non-contractible, we change its label to non-contractible. If it is contractible, we contract it and change the labels of up to four edges (the two tree edges above and the two tree edges below) to "unknown status". If we run out of edges with unknown status before n reaches four, the graph is not triconnected. If n reaches four, the graph is contracted to a K_4 and we have proved triconnectivity.

Lemma 8 *The algorithm performs at most $5n$ tests for contractability.*

Proof: Consider the potential function

$$5 \cdot \#\text{ of tree edges} + \#\text{ of edges labelled "unknown status".}$$

Its initial value is $5(n-1)$ and it decreases by at least one in every iteration. If the edge tested is contractible, the first term goes down by five and the second term goes up by at most four, and if the edge tested is non-contractible, the first term does not change and the second term decreases by one. The potential is always non-negative. We conclude that the number of iterations is bounded by $5n$. □

We next characterize the non-contractible tree edges. If a tree edge xy is non-contractible, there must be a z such that $\{x, y, z\}$ is a separation triple. The following lemma deals with the case $z < x$. A symmetric lemma deals with the case $z > y$.

Lemma 9 *Let xy with $y = x+1$ be a tree edge and let $z < x$. The triple $\{x, y, z\}$ splits G if and only if*

- *$z < x - 1$,*
- *$z > 1$ if $y = n$,*
- *there is no chord (u, v) with either*

  property one: *$u > y$ and $z < v < x$ or*

  property two: *$z < u < x$ and $v < z$*

Proof: If $\{x, x+1, z\}$ with $z < x$ is a separation triple, the two components consist of vertices $z+1, \ldots, x-1$ and $x+2, \ldots, z-1$, respectively; here we use the fact that $(n, 1)$ is among the back edges. Therefore, we must have $z+1 \leq x-1$ and $z > 1$ if $y = n$. Also, there can be no back edge connecting the two sets. □

We next specify data structures dealing with properties one and two. Consider a tree edge xy with $y = x + 1$. We query both data structures with x. The *data structure for property one* returns

$$z_1(x) := \max\{z < x \,|\, \text{there is a back edge } uz \text{ with } u > y\}.$$

So there is no back edge starting above y and ending between $z_1(x)$ and x. However, there is such a back edge $uz_1(x)$ with $u > y$. The *data structure for property two* yields

$$z_2(x) := \max\{z < x - 1 \,|\, \text{there is no back edge } uv \text{ with } v < z < u < x\}.$$

If $z_2(x) \geq z_1(x)$ (and $y < n$ or $z_2(x) > 1$), $\{x, y, z_2(x)\}$ is a separation triple, because there is no back edge having one endpoint in $\{z_2(x) + 1, \ldots, x - 1\}$ and one endpoint in $\{y + 1, \ldots, n, 1, \ldots, z_2(x) - 1\}$. If $z_2(x) < z_1(x)$, there is no separation triple $\{x, y, z\}$ with $z < x$. Assume otherwise. If $z < z_1(x)$, the back edge $uz_1(x)$ with $u > y$ witnesses that $\{x, y, z\}$ is not separating and if $z \geq z_1(x) > z_2(x)$, there is a back edge uv with $v < z < u < x$; it witnesses that $\{x, y, z\}$ is not separating.

We need to clarify how the graph is represented after a sequence of contractions. Each vertex of the current graph corresponds to a set of consecutive vertices of the original graph. We maintain a union-find data structure for the correspondence between original vertices and current vertices. For any vertex x of G, let $cur(x)$ be the vertex of the current graph containing x. With each vertex of the current graph, we store the highest and lowest original vertex contained in it. The tree edges of the current graph are tree edges of the original graph. The back edges $\mathcal{B}_c$ of the current graph are the edges $(cur(u), cur(v))$, where $uv \in \mathcal{B}$, $cur(v) \neq cur(u)$ and $(cur(v), cur(u))$ is not parallel to a tree edge of the current graph.

The running times of general solutions for the union-find problem are not quite linear. However, in the following situation the amortized cost of unions and finds is constant [IA87, GT85]. The union-find data structure is on a set of items and some forest, called the union-tree, on these items is specified when the union-find data structure is initialized. The blocks of the union-find data structure are subtrees of this forest. More precisely, forest edges are either light or solid. At the beginning all edges are light; the blocks are the connected components formed by solid edges. A union turns an edge from light to solid. For the union-find structure Cur, the union tree is the path $(1, 2, \ldots, n)$.

As shown before, computing whether a tree edge $e = (x, y)$ with $y = x + 1$ is contractible in G reduces to computing the vertices $z_1(x)$ and $z_2(x)$.

Computing $z_1(x)$: We store all back edges uv in a data structure for 2-dimensional orthogonal range queries [Meh84, dBKOS97]. The back edge uv is stored as the point (u, v). Let x_ℓ be the lowest numbered vertex in $cur(x)$ and let y_h be the highest numbered vertex in $cur(y)$. We determine

$$\max\{v < x_\ell \,|\, \text{there is a back edge } uv \in B \text{ with } u > y_h\}.$$

Then $cur(v)$ is the highest numbered vertex below $cur(x)$ in the current graph that has an incoming back edge from a source above $cur(y)$ in the current graph. Using

the results for 3-sided two-dimensional orthogonal range queries in [ABR00, Section 2.1, Theorem 4], which are based on computing iteratively nearest common ancestors in a Cartesian tree as shown by Bentley, Gabow and Tarjan [GBT84, Section 3], the query takes $O(1)$ time.

Computing $z_2(x)$: For any x, let

$$cand(x) = \{z < x \,|\, \text{there is no back edge } uv \text{ with } v < z < u < x\}.$$

We show how to compute the sets $cand(x)$ for all x in linear time. The following recursive characterization is useful: $cand(1) = \emptyset$ and for $x > 1$,

$$cand(x) = \{x-1\} \cup (cand(x-1) \setminus \{z \,|\, (x-1, v) \text{ is a back edge and } v < z\}),$$

i.e., we delete all z from $cand(x-1)$ that lie above the lowest back edge emanating from $x-1$, and add $x-1$. We can compute and store all sets $cand(x)$ in a forest in linear time and space (see Figure 1(a)), as vertices that are deleted once will not be considered again for the next cand-lists. For any x, the elements in $cand(x)$ lie on a leaf-to-root path in this forest. Assume we have already the list $cand(x-1)$. Let $(x-1, v)$ be the back edge starting in $x-1$ with smallest v. Then $cand(x)$ starts with a new item labelled $x-1$ and this item points to the first element on $cand(x-1)$ which is less than or equal to v (if it exists).

Given the *cand*-structure, it is easy to compute $z_2(x)$ by just taking the second element in $cand(x)$. But how does the *cand*-structure evolve as edges are contracted? The *cand*-structure for the current graph has one item for each vertex of the current graph. The vertices of the current graph correspond to the blocks of the *cur*-union-find structure and hence we will use these blocks as the vertices of the *cand*-structure for the current graph. Each block has an outgoing *cand*-edge. This edge goes to an item of the *cur*-structure. A find applied to this item then brings us to the block containing the item (= the true target of the *cand*-edge). Initially, all blocks are singletons, finds are trivial, and hence the structure is as described in the previous paragraph.

When we contract a tree edge (x, y), the vertices $cur(x)$ and $cur(y)$ are merged. Let us call the new vertex v_{xy}; after the union-operation on $cur(x)$ and $cur(y)$, we have $v_{xy} = cur(x) = cur(y)$. There are *cand*-edges (= edges in the *cand*-structure) to $cur(x)$ and $cur(y)$; they should now go to v_{xy}. This requires no action as the *cur*-union-find structure takes care of the indirection. How do the *cand*-sets change? Let us conceptually go through the construction of the *cand*-structure for the current graph and relate it to the *cand*-structure before the contraction. For the vertices below $cur(x)$ nothing changes (see Figure 1). Also, $cand(v_{xy})$ is simply $cand(cur(x))$.

Let us next consider the first vertex a above v_{xy} in the current graph (i. e., the first vertex above $cur(y)$ before the contraction). The first element in $cand(a)$ is v_{xy}; before the contraction it was $cur(y)$ and hence union-find takes care of the change. The second element is the first element on $cand(v_{xy})$ that is smaller or equal to the lowest back edge going out of v_{xy}. This back edge is the lowest back edge out of either $cur(x)$ or

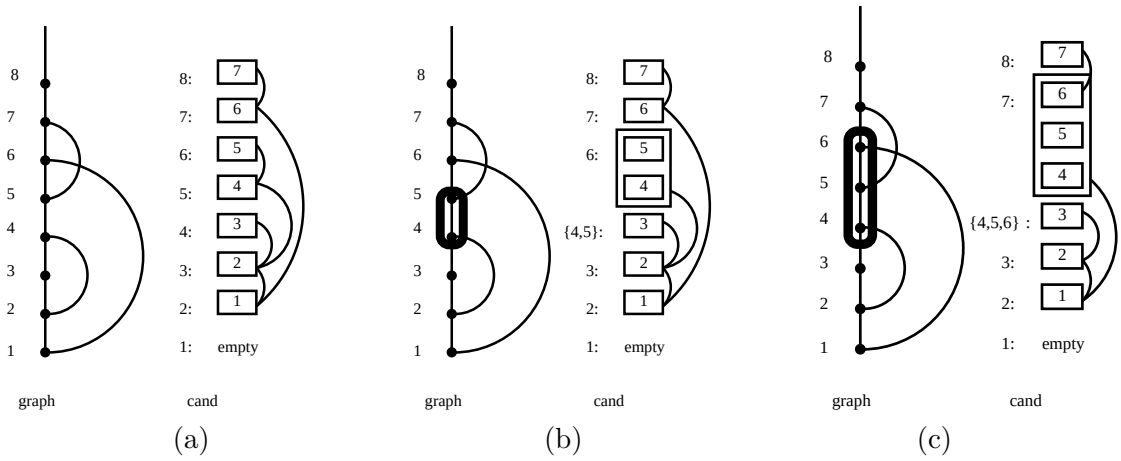

Figure 1: (a) shows the original graph and the corresponding *cand*-structure. All edges in the *cand*-structure are directed downwards; $cand(6)$ is equal to $\{5, 4, 2, 1\}$ and $cand(7)$ is equal to $\{6, 1\}$.
(b) shows the graph and the corresponding *cand*-structure after contracting the edge $(4, 5)$; $cand(6)$ is equal to $\{\{5, 4\}, 2, 1\}$ and $cand(7)$ is equal to $\{6, 1\}$.
(c) shows the graph and the corresponding *cand*-structure after contracting the edges $(4, 5)$ and $(5, 6)$; $cand(7)$ is equal to $\{\{6, 5, 4\}, 1\}$.

$cur(y)$ before the contraction. If neither $cur(x)$ nor $cur(y)$ contributes a back edge, the second element of $cand(a)$ after the contraction is the first element of $cand(v_{xy})$. If $cur(x)$ contributes the lowest back edge, the second element on $cand(a)$ after the contraction becomes the second element of $cand(cur(y))$ before the contraction, as shown in Figure 1(b). If $cur(y)$ contributes the lowest back edge, the second element of $cand(a)$ after the contraction is simply the second element of $cand(a)$ before the contraction, as shown in Figure 1(c). Finally, for the vertices above a, nothing changes, except that edges into $cur(x)$ or $cur(y)$ are now interpreted as edges in v_{xy}.

We conclude that the *cand*-structure can be updated in constant time after a contraction. We have now shown our main result.

Theorem 10 *Let G be a triconnected Hamiltonian graph with known Hamiltonian cycle. Then a Tutte contraction sequence for G can be constructed in linear time $O(n + m)$.*

4 Finding a Separation Pair

The algorithms of Hopcroft-Tarjan and Miller-Ramachandran find separation pairs in graphs that are not triconnected. In the case of Hamiltonian graphs, separation pairs can also be found using the data structures defined above.

Lemma 11 *Vertices x and z with $x > z$ form a separation pair if $z < x - 1$, either $z > 1$ or $x < n$, and there is no edge (u, v) with $z < u < x$ and $v < z$ or $v > x$.*

We can use our data structures for properties one and two to check for the existence of such a pair $\{x, z\}$ in linear time. Consider a fixed x. The data structure for property one yields the maximal $z < x$, call it $z_1(x)$, such that there is a back edge $uz_1(x)$ with $u > x$. The data structure for property two yields the maximal $z < x - 1$, call it $z_2(x)$, such that there is no back edge uv with $v < z < u < x$. If $z_2(x) \geq z_1(x)$, $\{x, z_2(x)\}$ is a separation pair. If $z_2(x) < z_1(x)$, there is no separation pair $\{x, z\}$ with $z < x$.

5 An Extension

We consider *extended Hamiltonian graphs*. Let C be a simple cycle. We can now add chords and segments. A *chord* is an edge whose endpoints are nonadjacent vertices on the cycle. A *segment* s consists of a new vertex $v(s)$ that is connected to three or more vertices on C. We feel that the generalization is interesting for two reasons: First, they are exactly the kind of graph considered in the merge step of the algorithm of Hopcroft and Tarjan. Second, we do not achieve linear time but only time $O(n \log n + m)$. So the generalization adds something qualitatively new and hints that the generalization of our results to general graphs will be non-trivial.

Let G be an extended Hamiltonian graph and let G' be a Hamiltonian graph obtained from G as follows. Number the vertices of C consecutively, starting at an arbitrary vertex. In this way, C becomes a path P plus a back edge. C and all the chords of G belong to G'. Let s be any segment of G and let $v_1 < v_2 < \ldots < v_k$ be its attachments on C. For each i, $1 \leq i \leq k$, we add the chords v_1v_i, $v_{i-1}v_i$, v_iv_{i+1} and v_iv_k, except if such a chord is a self-loop or connects adjacent vertices on P.

Lemma 12 *G is triconnected if and only if G' is triconnected.*

Proof: Assume first that G is not triconnected and let $\{a, b\}$ be a separation pair. Then a and b must belong to C. Thus $\{a, b\}$ splits G'.

Assume next that G' is not triconnected and let $\{a, b\}$ be a separation pair. $C \setminus \{a, b\}$ consists of two nonempty paths, say Q_1 and Q_2 and no chord of G' connects Q_1 and Q_2. Assume that there is a segment s of G that connects Q_1 and Q_2; let $v_1 < v_2 < \ldots < v_k$ be the attachments of s. Let v_i lie on Q_1 and let v_j lie on Q_2. We may assume $i < j$. If one of v_1 or v_k lies on Q_1 or Q_2, G' contains a chord that connects Q_1 and Q_2. Therefore $\{a, b\} = \{v_1, v_k\}$ and hence the vertices v_i to v_j are distinct from a and b. There must be an ℓ such that $v_\ell \in Q_1$ and $v_{\ell+1} \in Q_2$. Thus G' contains a chord connecting Q_1 and Q_2. □

In order to prove triconnectivity of G, we prove triconnectivity of G'. A segment s of G with k attachments gives rise to no more than $3k - 6$ edges of G'. Thus linear time in the size of G' is linear time in the size of G.

It is easy to extend a contraction sequence for G' into a contraction sequence for G. Let e_1, e_2, $\ldots e_{n-4}$ be the contraction sequence for G' where e_1 to e_{n-4} are distinct

tree edges. The contractions transform G' into K_4 without generating a vertex of degree two. We perform a slightly extended sequence of contractions in G. It suffices to explain the correspondence for the first contraction. A contraction of e_1 in G might generate a segment with only two attachments and hence a vertex of degree two. This is the case if $e_1 = xy$ and there is a segment in G having attachments x, y and z. The segment gives rise to edges xz and yz in G'; one of these edges is missing if z is a neighbor of x or y on C. Let $v(s)$ be the internal vertex of s. In G, we first contract the tree edge into $v(s)$ and then xy.

It remains to describe how we detect the segments having exactly three attachments and x and y among them. Again, we maintain a union-find data structure for the vertices of G'. The blocks correspond to the vertices of the current graph, i.e., two vertices of G' belong to the same block if they have been contracted into the same vertex of the current graph. For a vertex x, let $B(x)$ be the block containing x. With each such block B we maintain a (mergeable) priority queue PQ_B; it contains all segments s having an attachment in B and at least one attachment above B. The segments are ordered in increasing order by their first attachment above B.

Assume now we want to contract the tree edge xy with y being above x. We perform findmin operations on $PQ_{B(x)}$. Let s be the segment delivered and let y' be the next attachment of s above B. If $y' \in B(y)$, the contraction of xy merges two attachments of s into one. We decrease the degree of $v(s)$ and if the degree becomes two, we contract the tree edge into $v(s)$. We continue until $PQ_{B(x)}$ delivers a segment whose lowest attachment y' above B does not lie in $B(y)$. We unite $B(x)$ and $B(y)$ into a common block and we merge $PQ_{B(x)}$ and $PQ_{B(y)}$ into the priority queue for the new block.

Since mergeable priority queues can be maintained in time $O(\log n)$ per operation, we obtain:

Theorem 13 *A Tutte contraction sequence for an extended Hamiltonian graph can be constructed in time $O(n \log n + m)$.*

6 Checking the Witness

Let G be an arbitrary graph and let $e_1, e_2, \ldots, e_{n-4}$ be a sequence of edges of G. Let $G_0 = G$ and let $G_i = G_{i-1}/e_i$. We describe a linear-time algorithm that checks whether the sequence is a Tutte contraction sequence for G.

The question is how to maintain the graphs G_i? We use a union-find data structure B to represent the vertices of G_i. Each block is a vertex of G_i. The adjacency lists for each block contain vertices of the original graph, and may describe parallel edges and self-loops. According to Theorem 4, we need to verify that, before the contraction of an edge xy, x and y have at least three neighbors each. We describe the test for x. Let $B(x)$ be the block containing x. We iterate over the adjacency list for $B(x)$ and build up a set S of neighbors of $B(x)$. We start with S being the empty set and stop the scan as soon as S has three elements. Assume now that an edge $x'y'$ with $x' \in B(x)$ is scanned. If $y' \in B(x)$, $x'y'$ is a self-loop in the current graph and we simply delete it from the

adjacency list of $B(x)$. If $y' \notin B(x)$, but $B(y') \in S$, we are scanning an edge parallel to an edge scanned previously and we delete it from the adjacency list. If $y' \notin B(x)$ and $B(y') \notin S$, we add $B(y')$ to S. The scan over the adjacency list of $B(x)$ ends once the size of S reaches three. If we run out of edges before, $B(x)$ has degree less than three in the current graph and we reject the contraction sequence. The number of edges required to check is therefore three plus the number of edges deleted, amortizing to a total of $O(n + m)$ checks.

It remains to discuss the details of the union-find data structure. We remarked in Section 3 that there are linear-time solutions to the union-find problem if a union-tree prescribing the potential unions is part of the input. For the union-find structure B required in this section, the union tree is defined as follows. Let $T = (V, \{e_1, \ldots, e_{n-4}\})$. We first verify that T is a forest. If not, the sequence is rejected, because one of the contractions will contract a self-loop and hence the final graph will have more than four vertices. If T is a forest, it contains four trees, one for each vertex of the final graph. These trees build the prescribed set of possible unions and therefore yield linear total time.

We still need to check that the final graph is K_4. The graph has four vertices. As described in the preceding paragraph, we remove self-loops and parallel edges. Then we only need to check whether there are six edges left.

Theorem 14 *Given a graph G and a sequence e_1 to e_{n-4} of edges of G, one can check in linear time whether the sequence is a Tutte contraction sequence for G.*

One of the reviewers remarked that the linear-time solution for the union-find problem with given union-tree [GT85] is fairly complex and hence should not be used in an algorithm for checking witnesses. This criticism is valid; it can be overcome in two ways. First, one may sacrifice linear running time and resort to almost linear solutions of the general union-find problem. Second, one may require that the vertices of the graph are numbered in a special way; it would be an obligation of the triconnectivity algorithm to compute such a numbering. The numbering has the property that for all i, $1 \leq i \leq n-4$, the vertices in any component of $T = (V, \{e_1, \ldots, e_i\})$ are numbered consecutively. It is easy to compute such a numbering in linear time using the linear-time solution for the union-find problem with given union-tree. This numbering replaces the union tree by a union path, where possible unions are performed only between consecutive vertices. Then a much simpler approach of [IA87] applies.

7 Higher Connectivity

Connectivity and two-connectivity are easily certified in linear time. A spanning tree certifies connectivity, a cut vertex certifies non-two-connectivity, and every s-t-numbering, open ear decomposition and bipolar orientation certifies two-connectivity [Bra02].

For arbitrary k, the situation is as follows. In linear time any k-connected graph can be sparsified to a k-connected graph with $O(kn)$ edges [NI92]. Of course, certifying k-connectivity of the sparsification certifies k-connectivity of the original graph. Also, it is

easy to check whether a vertex cut in the sparsification is also a vertex cut in the original graph. k-connectivity can be tested in time $O(m + \min(kn^2, k^2n^{3/2}))$ ([Eve75, Gal80]); the algorithms are certifying. The certificates are flows.

[LLW88] gave a geometric characterization of k-connectivity for general k. A graph G is k-connected if and only if, specifying any k vertices of G, the vertices of G can be represented as points in R^{k-1} so that no k are on a hyperplane and each vertex is in the convex hull of its neighbors, except for the k specified vertices. The characterization gives rise to an $O(n^{5/2} + nk^{5/2})$ time Monte Carlo algorithm for k-connectivity and an $O(kn^{5/2} + nk^{7/2})$ Las Vegas algorithm for k-connectivity. The algorithms are certifying.

For $k \geq 4$, k-connected graphs do not necessarily contain edges that preserve k-connectivity upon contraction [Kri02] (we call such edges *k-contractible* edges). This is in sharp contrast to $k = 1, 2, 3$, in particular, to Theorem 1 for the case of 3-connected graphs. Although it is possible to characterize the non-complete 4-connected graphs that do not contain 4-contractible edges, an analogue characterization for $k = 5$ is regarded as tremendously hard problem in [Kri02]. However, some subclasses of k-connected graphs have been proven to contain a k-contractible edge, e. g., the class of triangle-free k-connected graphs [Tho81].

8 Conclusion

We gave a linear-time certifying triconnectivity algorithm for Hamiltonian graphs and an $O(n \log n + m)$ algorithm for extended Hamiltonian graphs. The challenge is to generalize this result to all graphs. We have also shown that the validity of a contraction sequence can be checked in linear time by a simple algorithm.

9 Acknowledgment

We want to thank the reviewers for their insightful comments on the original version of this paper.

References

[ABR00] S. Alstrup, G. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *FOCS*, pages 198–207, 2000.

[AES87] K. Ando, H. Enomoto, and A. Saito. Contractible edges in 3-connected graphs. *Journal of Combinatorial Theory, Series B*, 42:87–93, 1987.

[Bra02] U. Brandes. Eager st-ordering. In *ESA*, pages 247–256, 2002.

[dBKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 1997.

[EMS10] A. Elmasry, K. Mehlhorn, and J. M. Schmidt. Every DFS-Tree of a 3-Connected Graph Contains a Contractible Edge. Available at the second author's web-page, February 2010.

[Eve75] Shimon Even. An algorithm for determining whether the connectivity of a graph is at least k. *SIAM Journal on Computing*, 4(3):393–396, 1975.

[Gal80] Zvi Galil. Finding the vertex connectivity of graphs. *SIAM Journal on Computing*, 9(1):197–199, 1980.

[GBT84] H. Gabow, J. Bentley, and R. Tarjan. Scaling and related techniques for geometry problems. In *STOC*, pages 135–143, 1984.

[GM00] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In *Graph Drawing*, volume 1984 of *LNCS*, pages 77–90, 2000.

[GT85] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–221, 1985.

[HT74] J. Hopcroft and R. E. Tarjan. Efficient Planarity Testing. *Journal of the ACM*, 21(4):549–568, 1974.

[IA87] T. Imai and T. Asano. Dynamic segment intersection with applications. *Journal of the ACM*, 8:1–18, 1987.

[Kri02] M. Kriesell. A survey on contractible edges in graphs of a prescribed vertex connectivity. *Graphs and Combinatorics*, 18(1):1–30, 2002.

[LLW88] N. Linial, L. Lovász, and A. Wigderson. Rubber bands, convex embeddings, and graph connectivity. *Combinatorica*, 8(1):91–102, 1988.

[Meh84] K. Mehlhorn. *Data Structures and Efficient Algorithms*, volume 3: Multi-dimensional Searching and Computational Geometry. Springer, 1984. 284 pages.

[MMNS10] R. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying Algorithms. to appear in Computer Science Review, June 2010.

[MR92] G. L. Miller and V. Ramachandran. A new graph triconnectivity algorithm and its parallelization. *Combinatorica*, 12(1):53–76, 1992.

[NI92] H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7:583–596, 1992.

[Sch10] J. M. Schmidt. Construction sequences and certifying 3-connectedness. In *27th International Symposium on Theoretical Aspects of Computer Science (STACS'10)*, Nancy, France, 2010. `http://drops.dagstuhl.de/portals/extern/index.php?conf=STACS10`.

[Tho81] C. Thomassen. Nonseparating cycles in k-connected graphs. *J. Graph Theory*, 5:351–354, 1981.

[Tut61] W. Tutte. A theory of 3-connected graphs. *Indag. Math.*, 23:441–455, 1961.