On the complexity of computing treelength

Daniel Lokshtanov*

Abstract

We resolve the computational complexity of determining the *treelength* of a graph, thereby solving an open problem of Dourisboure and Gavoille, who introduced this parameter, and asked to determine the complexity of recognizing graphs of bounded treelength [6]. While recognizing graphs with treelength 1 is easily seen as equivalent to recognizing chordal graphs, which can be done in linear time, the computational complexity of recognizing graphs with treelength 2 was unknown until this result. We show that the problem of determining whether a given graph has treelength at most k is NP-complete for every fixed $k \geq 2$, and use this result to show that treelength in weighted graphs is hard to approximate within a factor smaller than $\frac{3}{2}$. Additionally, we show that treelength can be computed in time $O^*(1.7549^n)$ by giving an exact exponential time algorithm for the Chordal Sandwich problem and showing how this algorithm can be used to compute the treelength of a graph.

1 Introduction

Treelength is a graph parameter proposed by Dourisboure and Gavoille [6] that measures how close a graph is to being chordal. The treelength of G is defined using tree decompositions of G. Graphs of treelength k are the graphs that have a tree decomposition where the distance in G between any pair of nodes that appear in the same bag of the tree decomposition is at most k. As chordal graphs are exactly those graphs that have a tree decomposition where every bag is a clique [16], [3], [11], we can see that treelength generalizes this characterization.

There are several reasons for why it is interesting to study this parameter. For example, Dourisboure et. al. show that graphs with bounded treelength have sparse additive spanners [5]. Dourisboure also shows that graphs of bounded treelength admit compact routing schemes [4]. One should also note that many graph classes with unbounded treewidth have bounded treelength, such as chordal, interval, split, AT-free, and permutation graphs [6].

In this paper, we show that recognizing graphs with treelength bounded by a fixed constant $k \geq 2$ is NP-complete. The problem of settling the complexity of recognizing graphs of bounded treelength was first posed as an open problem

*Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Email: daniel.lokshtanov@uib.no.

by Dourisboure and Gavoille, and remained open until this result [6]. Our result is somewhat surprising, because by bounding the treelength of G we put heavy restrictions on the distance matrix of G. Another indication that this problem might be polynomial for fixed k was that the treelength of a graph is fairly easy to approximate within a factor of 3 [6]. In comparison, the best known approximation algorithm for treewidth has an approximation factor of $O(\sqrt{\log k})$ [7]. As Bodlaender showed, recognizing graphs with treewidth bounded by a constant can be done in linear time [1]. Since the above observation about approximability might indicate that determining treelength is "easier" than determining treewidth, one could arrive at the conclusion that recognizing graphs with treelength bounded by a constant should be polynomial. However, there are also strong arguments against this intuition. For instance, graphs of bounded treelength are just bounded diameter graphs that have been glued together in a certain way. Thus, when trying to show that a graph indeed has small treelength one would have to decompose the graph into components of small diameter and show how these components are glued together to form the graph. As the class of bounded diameter graphs is very rich, one would have a myriad of candidates to be such components, making it hard to pick out the optimal ones. This intuition is confirmed when we prove the hardness of recognizing graphs of bounded treelength because the instances we reduce to all have bounded diameter.

In the next section we will give some notation and preliminary results. Next, we present a proof that determining whether the treelength of a weighted graph is less than or equal to k is NP-hard for every fixed $k \geq 2$. Following this, we reduce the problem of recognizing weighted graphs with treelength bounded by k to the problem of recognizing unweighted graphs with the treelength bounded by the same constant k, thereby completing the hardness proof. Finally we also consider the complexity of approximating treelength, and propose a fast exact algorithm to determine the parameter by solving the Chordal Sandwich problem.

2 Notations, terminology and preliminaries

We employ $O^*()$ notation for which suppresses polynomial factors. Formally $f(n) = O^*(g(n))$ if there is a constant c such that $f(n) = O(g(n)n^c)$. For a graph $G = (V, E)$ let $w : E \to \mathbb{N}$ be a *weight function* on the edges. The *length* of a path with respect to a weight function w is the sum of the weights of its edges. The *distance* $d_w(u, v)$ between two vertices is the length of the shortest path with respect to w. Whenever no weight function is specified the unit weight function $w(e) = 1$ for all $e \in E$ is used. G to the *power* of k with respect to the weight function w is $G^k_w = (V, \{uv : d_w(u, v) \leq k\})$. A weight function w is *metric* if it satisfies a generalization of the triangle inequality, that is, if $w((u, v)) = d_w(u, v)$ for every edge (u, v).

A *tree decomposition* of a graph $G = (V, E)$ is a pair (S, T) consisting of a set $S = \{X_i : i \in I\}$ of *bags* and a tree $T = (I, M)$ so that each bag $X_i \in S$ is a subset of V and the following conditions hold:

- $\bigcup_{i \in I} X_i = V$
- For every edge (u, v) in E, there is a bag X_i in S so that $u \in X_i$ and $v \in X_i$
- For every vertex v in V, the set $\{i \in I : v \in X_i\}$ induces a connected subtree of T

The *length* of a bag is the maximum distance in G between any pair of vertices in the bag. The *length* of a tree-decomposition is the maximum length of any bag. The *treelength* of G with weight function w is the minimum length of a tree-decomposition of G, and is denoted by $tl_w(G)$. When no weight function is given on E, then the treelength is denoted by $tl(G)$. A *shortest* tree decomposition is a tree decomposition having minimum length. We will always assume that all weight functions are metric. This can be justified by the fact that if w is not metric, we easily can make a new metric weight function w' by letting $w'((u, v)) = d_w(u, v)$ for every edge (u, v) and observe that $tl_{w'}(G) = tl_w(G)$.

The *neighbourhood* of a vertex v is denoted $N(v)$ and is the vertex set $\{u : (u, v) \in E\}$. When S is a subset of V, $G[S] = (S, E \cap \{(u, v) : u \in S, v \in S\})$ is the subgraph *induced* by S. We will use $G \setminus v$ to denote the graph $G[V \setminus \{v\}]$. G is *complete* if (u, v) is an edge of G for every pair $\{u, v\}$ of distinct vertices in G. A *clique* in G is a set S of vertices in G so that $G[S]$ is complete.

For two graphs $G = (V, E)$ and $G' = (V, E')$, $G \subseteq G'$ means that $E \subseteq E'$. For a graph class Π, G is a Π-*graph* if $G \in \Pi$. G' is a Π-*sandwich* between G and G'' if G' is a Π-graph and $G \subseteq G' \subseteq G''$ [12]. A graph class Π is *hereditary* if every induced subgraph of a Π-graph is a Π-graph.

A graph is *chordal* if it contains no induced cycle of length at least 4. Thus the class of chordal graphs is hereditary. A vertex v is *simplicial* if the neighbourhood of v is a clique. A vertex v is *universal* if $V = \{v\} \cup N(v)$. An ordering of the vertices of G into $\{v_1, v_2, \ldots, v_n\}$ is a *perfect elimination ordering* if for every i, v_i is simplicial in $G[\{v_j : j \geq i\}]$. A *clique tree* of G is a tree decomposition of G such that every bag is a maximal clique of G (see e.g., [13] for details).

Theorem 2.1 *The following are equivalent:*

- *G is chordal.*
- *G has a clique tree. [16], [3], [11]*
- *G has a perfect elimination ordering. [10]*

For more characterizations of chordal graphs and the history of this graph class, refer to the survey by Heggernes [13]. Following Theorem 2.1 it is easy to see that if v is simplicial then G is chordal if and only if $G \setminus v$ is chordal. Universal vertices share this property, as has been observed by several authors before.

Observation 2.2 *If v is universal in G then G is chordal if and only if $G \setminus v$ is chordal.*

Proof. If G is chordal then $G \setminus v$ is chordal because the class of chordal graphs is hereditary. Now suppose $G \setminus v$ is chordal. Consider a perfect elimination ordering of $G \setminus v$ appended by v. This is clearly a perfect elimination ordering of G, hence G is chordal. ∎

We now define the problem that we are going to show is NP-complete. In the problem statement below, k is an integer greater than or equal to 2.

k-Treelength
Instance: A graph G
Question: Is $tl(G) \leq k$?

Finally, we define the problem we will reduce from.

Chordal Sandwich [12]
Instance: Two graphs G_1 and G_2 with $G_1 \subseteq G_2$
Question: Is there a chordal sandwich between G_1 and G_2?

3 Weighted k-Treelength is NP-Complete

In this section we are going to show that determining whether the treelength of a given weighted graph is at most k is NP-complete for every fixed $k \geq 2$. In the next section we will conclude the hardness proof for unweighted graphs by showing how one from a weighted graph G in polynomial time can construct an unweighted graph G' with the property that $tl_w(G) \leq k$ if and only if $tl(G') \leq k$.

Weighted k-Treelength
Instance: A graph G with weight function w
Question: Is $tl_w(G) \leq k$?

Observation 3.1 *For a graph $G = (V, E)$ and a weight function w, $tl_w(G) \leq k$ if and only if there exists a chordal sandwich G' between G and G^k_w.*

Proof. Suppose $tl_w(G) \leq k$. Consider a shortest tree decomposition (S, T) of G, and construct the graph $G' = (V, \{(u, v) : \exists i\ u \in X_i, v \in X_i\})$. $G \subseteq G'$ is trivial, $G' \subseteq G^k_w$ holds because the length of the tree decomposition is at most k, and G' is chordal because (S, T) is a clique tree of G'. In the other direction, let G' be a chordal sandwich between G and G^k_w. Consider a clique tree (S, T) of G'. This is a tree decomposition of G, and the length of this decomposition is at most k, as $u \in X_i$ and $v \in X_i$ implies $(u, v) \in E(G') \subseteq E(G^k_w)$. ∎

Corollary 3.2 *For any graph G, $tl(G) = 1$ if and only if G is chordal.*

From Observation 3.1, it follows that determining the treelength of a given graph in fact is a special case of the *Chordal Sandwich* problem defined above. In a study of sandwich problems [12], Golumbic et. al. point out that as a consequence of the hardness of Triangulating Colored Graphs, the Chordal Sandwich problem is NP-Complete. Thus, in order to prove that Weighted k-Treelength is indeed hard, we only need to reduce the Chordal Sandwich problem to a special case of itself, namely the one where $G_2 = {G_1}^k_w$ for some weight function w.

We will reduce in the following way. On input $G_1 = (V_1, E_1)$, $G_2 = (V_1, E_2)$ with $G_1 \subseteq G_2$ to the Chordal Sandwich problem, let $E_D = E_2 \setminus E_1$. Observe that E_D is the set of possible fill edges we can add to G_1 in order to make it chordal. We construct a new graph G by taking a copy of G_1, adding a new vertex c_{uv} for every edge (u, v) in E_D and making this vertex adjacent to all other vertices of G. We denote the set of added vertices by C, as C is a clique of universal vertices. The weight function is simple, $w(c_{uv}, u) = w(c_{uv}, v) = \lfloor k/2 \rfloor$ for every c_{uv} and $w(e) = k$ for all other edges of G.

Lemma 3.3 *Let G, G_1 and G_2 be as described above. Then $tl_w(G) \leq k$ if and only if there is a chordal sandwich G' between G_1 and G_2.*

Proof. Observe that any supergraph G' of G on the same vertex set ($G' \supseteq G$) is chordal if and only if $G'[V_1]$ is chordal since every vertex in C is universal. Also, notice that for every pair u,v of vertices in V_1, $d_G(u, v) \leq k$ if and only if (u, v) is an edge of G_2. Thus it follows that $G_2 = G^k_w[V_1]$. Hence, by Observation 3.1, $tl_w(G) \leq k$ if and only if there is a chordal sandwich G' between G and G^k_w. By the discussion above, this is true if and only if there is a chordal sandwich between $G[V_1] = G_1$ and $G_2 = G^k_w[V_1]$. ∎

Corollary 3.4 *Weighted k-Treelength is NP-complete for every $k \geq 2$.*

Proof. By Lemma 3.3 determining whether a given weighted graph G has $tl_w(G) \leq k$ is NP-hard for every $k \geq 2$. By Observation 3.1 this problem is polynomial time reducible to the Chordal Sandwich problem, thus it is in NP. ∎

4 *k*-Treelength is NP-Complete

We will now show how one from a weighted graph G in polynomial time can construct an unweighted graph G'' with the property that $tl_w(G) \leq k$ if and only if $tl(G'') \leq k$. We do this in two steps. First we show how to construct a graph G' and weight function w' from G and w so that $tl_w(G) \leq k$ if and only if $tl_{w'}(G') \leq k$ and $w'(e) = 1$ or $w'(e) = k$ for every edge e in G'. In the second step we show how G'' can be constructed from G' and w'. Both steps are done in an inductive way. Obviously, if G has an edge of weight larger than k then $tl_w(G) > k$. We will therefore assume that $w(e) \leq k$ for all edges e. For an edge (u, v), let $G(u, v) = (V \cup \{r, q\}, (E \setminus (u, v)) \cup \{(u, r),(r, v),(u, q),(q, v)\})$. That is, we build

$G(u,v)$ from G by removing the edge (u,v), adding two new vertices r and q and making both of them adjacent to u and v. Let $w_{u,v,k}$ be a weight function of $G(u,v)$ so that $w_{(u,v,k)}(e) = w(e)$ if $e \in E(G) \cap E(G(u,v))$, $w_{(u,v,k)}((u,r)) = w_{(u,v,k)}((r,v)) = k$, $w_{(u,v,k)}((u,q)) = w(u,v) - 1$, and $w_{(u,v,k)}((q,v)) = 1$. Observe that if $w((u,v)) > 1$ then $w_{(u,v,k)}$ is properly defined.

Lemma 4.1 *Given a graph G, an edge (u,v), and a weight function w with $w((u,v)) > 1$, there is a chordal sandwich between G and G_w^k if and only if there is a chordal sandwich between $G(u,v)$ and $G(u,v)^k_{w_{(u,v,k)}}$.*

Proof. Suppose there is a chordal sandwich $\hat{G}_{(u,v)}$ between $G(u,v)$ and $G(u,v)^k_{w_{(u,v,k)}}$. The set $\{r,u,q,v\}$ induces a cycle in G and hence either (r,q) or (u,v) must be in edge in $E(\hat{G}_{(u,v)})$. The distance between r and q in $G(u,v)$ is $k+1$ we conclude that (u,v) must be in $E(\hat{G}_{(u,v)})$. Thus $\hat{G}_{(u,v)} \setminus \{r,q\}$, where r and q are the vertices that were added to $G \setminus (u,v)$ to obtain $G(u,v)$, is a chordal sandwich between G and G_w^k. In the other direction, suppose there is a chordal sandwich $\hat{G}$ between G and G_w^k. Then $\hat{G}' = (V(\hat{G}) \cup \{r,q\}, E(\hat{G}) \cup \{(u,r),(r,v),(u,q),(q,v)\})$ is a chordal sandwich between $G(u,v)$ and $G(u,v)^k_{w_{(u,v,k)}}$ because both r and q are simplicial nodes in $\hat{G}'$. ∎

Now, the idea is that the graph $G(u,v)$ with weight function $w_{(u,v,k)}$ is somewhat closer to not having any edges with weight between 2 and $k-1$. With an appropriate choice of measure, it is easy to show that this is indeed the case. The measure we will use will essentially be the sum of the weights of all edges that have edge weights between 2 and $k-1$. In the following discussion, let $W_w(G) = \sum_{e \in E, w(e) < k} (w(e) - 1)$. Observe that if $1 < w(u,v) < k$ then $W_{w_{(u,v,k)}}(G(u,v)) = W_w(G) - 1$, and that if $W_w(G) = 0$ then $w(e) = 1$ or $w(e) = k$ for every edge $e \in E$.

Lemma 4.2 *For a graph G with weight function w, we can construct in polynomial time a graph G' with weight function w' so that $|V(G')| = |V(G)| + 2W_w(G)$, and $tl_w(G) \leq k$ if and only if $tl_{w'}(G') \leq k$.*

Proof. We prove by induction on $W_w(G)$. If $W_w(G) = 0$ we know that $w(e) = 1$ or $w(e) = k$ for every edge e. Now, suppose the statement of the lemma holds for all graphs with $W_w(G) < t$ for some t and consider a graph G with weight function w so that $W_w(G) = t > 0$. Then, let (u,v) be an edge so that $1 < w((u,v)) < k$. By Lemma 4.1, $tl_w(G) \leq k$ if and only if $tl_{w_{(u,v,k)}}(G(u,v)) \leq k$. Now, $W_{w_{(u,v,k)}}(G(u,v)) = W_w(G) - 1$. Thus, by the induction assumption, we can in polynomial time construct a graph G' with weight function w' that satisfies $tl_w(G) \leq k \iff tl_{w_{(u,v,k)}}(G(u,v)) \leq k \iff tl_{w'}(G') \leq k$ with $|V(G')| = |V(G(u,v))| + 2W_{w_{(u,v,k)}}(G(u,v)) = |V(G)| + 2 + 2(W_w(G) - 1) = |V(G)| + 2W_w(G)$. ∎

The idea of the above proof is that we can use edges of weight 1 and k to emulate the behaviour of edges with other weights. The method we now will use to prove the hardness of unweighted treelength will be similar - we are going

to show that weight k edges can be emulated using only edges with weight 1. In order to do this, we are going to use the following lemma by Dourisboure et. al. concerning the treelength of cycles.

Lemma 4.3 *[6] The treelength of a cycle on k vertices is $\lceil \frac{k}{3} \rceil$.*

For an edge $(u, v) \in E$, we construct the graph $G[u, v, k]$ in the following way: We replace the edge (u, v) by three paths on $2k-1$, $2k-1$ and $k-1$ vertices respectively. Construct these paths $P_a = \{a_1, a_2, \ldots, a_{2k-1}\}$, $P_b = \{b_1, b_2, \ldots, b_{2k-1}\}$ and $P_c = \{c_1, c_2, \ldots, c_{k-1}\}$ using new vertices. Take a copy of G, remove the edge (u, v) and add edges from u to a_1, b_1 and c_1, and from v to a_{2k-1}, b_{2k-1} and c_{k-1}. For a weight function w of G, $w_{[u,v,k]}$ will be a weight function of $G[u, v, k]$ so that $w_{[u,v,k]}(e) = w(e)$ if $e \in E(G)$ and $w_{[u,v,k]} = 1$ otherwise.

Lemma 4.4 *Given G, weight function w and an edge $(u, v) \in E$ with $w(u, v) = k$, $tl_w(G) \leq k$ if and only if $tl_{w_{[u,v,k]}}(G[u, v, k]) \leq k$.*

Proof. Suppose there is a chordal sandwich $\hat{G}$ between G and G^k_w. We build $\hat{G}'$ from G by taking a copy of $\hat{G}$, adding three new paths $P_a = \{a_1, a_2, \ldots, a_{2k-1}\}$, $P_b = \{b_1, b_2, \ldots, b_{2k-1}\}$ and $P_c = \{c_1, c_2, \ldots, c_{k-1}\}$ and the edge sets $\{(u, a_i) : i \leq k\}$, $\{(u, b_i) : i \leq k\}$, $\{(u, c_i) : i \leq \lfloor \frac{k}{2} \rfloor\}$, $\{(v, a_i) : i \geq k\}$, $\{(v, b_i) : i \geq k\}$, $\{(v, c_i) : i \geq \lfloor \frac{k}{2} \rfloor\}$. We see that $\hat{G}'$ is chordal because $\{a_1, a_2 \ldots a_{k-1}$, $a_{2k-1}, a_{2k-2}, \ldots a_k, b_1, b_2 \ldots b_{k-1}$, b_{2k-1}, b_{2k-2}, $\ldots b_k$, c_1, c_2, $\ldots c_{\lfloor \frac{k}{2} \rfloor - 1}$, c_{k-1}, c_{k-2}, $\ldots c_{\lfloor \frac{k}{2} \rfloor}\}$ followed by a perfect elimination ordering of $\hat{G}$ is a perfect elimination ordering of $\hat{G}'$. Also, $\hat{G}' \subseteq G[u, v, k]^k_{w_{[u,v,k]}}$. Thus $\hat{G}'$ is a chordal sandwich between $G[u, v, k]$ and $G[u, v, k]^k_{w_{[u,v,k]}}$. In the other direction, let $\hat{G}_{[u,v]}$ be a chordal sandwich between $G[u, v, k]$ and $G[u, v, k]^k_{w_{[u,v,k]}}$. It is sufficient to show that $(u, v) \in E(\hat{G}_{[u,v]})$ because then $\hat{G}_{[u,v]}[V(G)]$ is a chordal sandwich between G and G^k_w. Consider the set $V_s = \{u, v\} \cup V(P_a) \cup V(P_b)$, and let C be the subgraph of $G[u, v, k]$ induced by V_s. Now, observe that $E(G[u, v, k]^k_{w_{[u,v,k]}}[S]) = E(C^k) \cup \{(u, v)\}$. Suppose for contradiction that (u, v) is not an edge of $\hat{G}_{[u,v]}$. Then we know that $\hat{G}_{[u,v]}[V_s]$ is a chordal sandwich between C and C^k implying that $tl(C) \leq k$. This contradicts Lemma 4.3 because C is a cycle on $4k$ vertices. ∎

Lemma 4.4 gives us a way to emulate edges of weight k using only edges of weight 1. For a graph G with weight function w, let $W_w[G] = |\{e \in E(G) : w(e) = k\}|$. Notice that if $w((u, v)) = k$ then $W_w[G] = W_{w_{[u,v,k]}}[G[u, v, k]] + 1$.

Lemma 4.5 *For every graph G with weight function w satisfying $w(e) = 1$ or $w(e) = k$ for every edge, we can construct in polynomial time a graph G' with $W_w[G](5k-3) + |V(G)|$ vertices and satisfying $tl_w(G) \leq k$ if and only if $tl(G') \leq k$.*

Proof. We use induction on $W_w[G]$. If $W_w[G] = 0$ the lemma follows immediately. Now, assume the result holds for $W_w[G] < t$ for some $t > 0$. Consider a graph G with weight function w so that $W_w[G] = t$. By Lemma 4.4 $tl_w(G) \leq k$ if and only if $tl_{w_{[u,v,k]}}(G[u,v,k]) \leq k$. By the inductive hypothesis we can construct in polynomial time a graph G' with $W_{w_{[u,v,k]}}[G[u,v,k]](5k-3) + |V(G[u,v,k])| + 5k - 3 = W_w[G](5k-3) + |V(G)|$ vertices and satisfying $tl(G') \leq k \iff tl_{w_{[u,v,k]}}(G[u,v,k]) \leq k \iff tl_w(G) \leq k$. ∎

Corollary 4.6 *For a graph G and weight function w, we can in polynomial time construct a graph G'' so that $tl_w(G) \leq k$ if and only if $tl(G'') \leq k$.*

Proof. By Lemma 4.2 we can in polynomial time construct a graph G' with weight function w' so that $tl_{w'}(G') \leq k \iff tl_w(G) \leq k$ and so that $w'(e) = 1$ or $w'(e) = k$ for every edge e in $E(G')$. By Lemma 4.5 we can from such a G' and w' construct in polynomial time a G'' so that $tl(G'') \leq k \iff tl_{w'}(G') \leq k \iff tl_w(G) \leq k$. ∎

Theorem 4.7 *Determining whether $tl(G) \leq k$ for a given graph G is NP-complete for every fixed $k \geq 2$.*

Proof. By Corollary 4.6, k-Treelength is NP-hard. As it is a special case of Weighted k-Treelength it is also NP-complete. ∎

5 Treelength is hard to approximate

Having established that treelength is hard to compute, it is natural to ask how well this parameter can be approximated. We say that a polynomial time algorithm that computes a tree-decomposition of G is a c-approximation algorithm for treelength if there is an integer k so that on any input graph G, the length l of the tree-decomposition returned by the algorithm satisfies the inequality $l \leq c \cdot tl(G) + k$. Dourisboure and Gavoille have already given a 3-approximation algorithm for treelength [6], and have conjectured that the parameter is approximable within a factor 2. In this section we show that as a consequence of the results in the above section, treelength in weighted graphs can not be approximated within a factor $c < \frac{3}{2}$ unless $P = NP$. For the treelength of unweighted graphs we give a weaker inapproximability result, and conjecture that there is no c-approximation algorithm for treelength with $c < \frac{3}{2}$ unless $P = NP$.

Lemma 5.1 *If $P \neq NP$ then, for any $c < \frac{3}{2}$, there is no polynomial time algorithm that on an input graph G returns a tree-decomposition of G with length $l \leq c \cdot tl(G)$.*

Proof. Suppose there is such an algorithm ALG. We give a polynomial time algorithm for 2-treelength, thereby showing that $P = NP$. On input G, run ALG on G, and let l be the length of the tree-decomposition of G returned by

ALG. Answer "$tl(G) \leq 2$" if $l < 3$ and "$tl(G) > 2$" otherwise. We now need to show that $tl(G) \leq 2$ if and only if $l < 3$. Assume $l < 3$. Then $tl(G) \leq l \leq 2$ as l is an integer. In the other direction, assume $tl(G) \leq 2$. In this case $l \leq c \cdot tl(G) < 3$. ∎

Unfortunately, Lemma 5.1 is not sufficient to prove that it is hard to approximate treelength within a factor $c < \frac{3}{2}$. The reason for this is that an algorithm that guarantees that $l \leq \frac{4}{3}tl(G) + 1$ can not be used to recognize graphs with treelength at most 2 in the above manner. However, we can show that there can be no c-approximation algorithms for the treelength of weighted graphs by using the weights on the edges to "scale up" the gap between 2 and 3.

Theorem 5.2 *If $P \neq NP$ then there is no polynomial time c-approximation algorithm for weighted treelength for any $c < \frac{3}{2}$.*

Proof. The proof is similar to the one for Lemma 5.1. Suppose there is a polynomial time c-approximation algorithm *ALG* for weighted treelength of G, with $c < \frac{3}{2}$. Let k be a non-negative integer so that on any graph G with weight function w, *ALG* computes a tree-decomposition of G with length $l < c \cdot tl(G) + k$. Now, choose t to be the smallest positive integer so that $(\frac{3}{2} - c) \cdot t \geq k + 1$. Let w be a weight function on the edges of G so that for every edge (u, v), $w((u, v)) = t$. Observe that $tl_w(G) = tl(G) \cdot t$. Run *ALG* on input (G, w) and let l be the length with respect to w of the tree-decomposition returned by *ALG*. Answer "$tl(G) \leq 2$" if $l < 3t$ and "$tl(G) > 2$" otherwise. We now need to show that $tl(G) \leq 2$ if and only if $l < 3t$. Assume $l < 3t$. Now, $tl(G) \cdot t = tl_w(G) \leq l < 3t$. Dividing both sides by t yields $tl(G) < 3$ implying $tl(G) \leq 2$ as $tl(G)$ is an integer. In the other direction, assume $tl(G) \leq 2$. In this case $l \leq c \cdot tl_w(G) + k = c \cdot tl(G) \cdot t + k = \frac{3}{2} \cdot tl(G) \cdot t - (\frac{3}{2} - c) \cdot tl(G) \cdot t + k \leq 3t - (k + 1) + k < 3t$. This implies that the described algorithm is a polynomial time algorithm for 2-Treelength implying $P = NP$. ∎

In fact, it does not seem that treelength should be significantly harder to compute on weighted than unweighted graphs. The hardness proof for k-Treelength is a reduction directly from weighted k-Treelength. Also, the exact algorithm given in the next section works as well for computing the treelength in weighted as in unweighted graphs. We feel that together with Lemma 5.1 and Theorem 5.2 this is strong evidence to suggest that unless $P = NP$, treelength is inapproximable within a factor $c < \frac{3}{2}$, also in unweighted graphs.

6 An exact algorithm for the Chordal Sandwich problem

In this section we give an exact algorithm that solves the Chordal Sandwich problem. The running time of this algorithm is $O^*(1.7549^n)$. In fact, the algorithm can be obtained by a quite simple modification of an exact algorithm to compute treewidth and minimum fill in given by Fomin et. al [8], together

with the tighter bound for the number of potential maximal cliques given by Fomin and Villanger [9]. Together with Observation 3.1 this gives a $O^*(1.7549^n)$ algorithm to compute the treelength of a graph. The algorithm applies dynamic programming using a list of the input graph's minimal separators and potential maximal cliques.

In order to state and prove the results in this section, we need to introduce some notation and terminology. Given two vertices u and v of G, a *minimal u-v-separator* is an inclusion minimal set $S \subseteq V$ so that u and v belong to distinct components of $G \setminus S$. A *minimal separator* is a vertex set S that is a minimal u-v-separator for some vertices u and v. We call a chordal supergraph H of G for a *minimal triangulation* of G if the only chordal sandwich between G and H is H itself. If $C \subseteq V$ is a maximal clique in some minimal triangulation of G, we say that C is a *potential maximal clique* of G. The set of all minimal separators of G is denoted by $\Delta(G)$ and the set of all potential maximal cliques is denoted by $\Pi(G)$. By $\mathcal{C}_G(S)$ we will denote the family of the vertex sets of the connected components of $G \setminus S$. Thus, if the connected components of $G \setminus S$ are $G[C_1]$ and $G[C_2]$, $\mathcal{C}_G(S) = \{C_1, C_2\}$. A *block* is a pair (S, C) where $S \in \Delta(G)$ and $C \in \mathcal{C}_G(S)$. A block is called *full* if $S = N(C)$. For a block (S, C) the realization of that block is denoted by $R(S, C)$ and is the graph obtained from $G[S \cup C]$ by making S into a clique.

The following two results are crucial for our proofs.

Lemma 6.1 *[14] Let S be a minimal separator of G. For every $C_i \in \mathcal{C}_G(S)$, let H_i be a minimal triangulation of $R(S, C_i)$. Then the graph H with $V(H) = V(G)$ and $E(H) = \bigcup_{C_i \in \mathcal{C}(S)} E(H_i)$ is a minimal triangulation of G. Conversely, let H be a minimal triangulation of G and let S be a minimal separator of H. Then, for every $C_i \in \mathcal{C}_G(S)$, $H[S \cup C_i]$ is a minimal triangulation of $R(S, C_i)$.*

Theorem 6.2 *[2] Let H be a minimal triangulation of G and let Ω be a maximal clique of H. Then for each block $(N(C_i), C_i))$ with $C_i \in \mathcal{C}_H(\Omega)$, $H[N(C_i), C_i]$ is a minimal triangulation of $R(N(C_i), C_i)$. Conversely, let Ω be a potential maximal clique of G. For each block $(N(C_i), C_i))$ with $C_i \in \mathcal{C}_G(\Omega)$ let H_i be a minimal triangulation of $R(N(C_i), C_i)$. Let H be a graph with $V(H) = V(G)$ and $E(H) = \{(u, v) : \{u, v\} \subseteq \Omega\} \cup \bigcup_{C_i \in \mathcal{C}_G(\Omega)} E(H_i)$. Then H is a minimal triangulation of G.*

We are now in position to prove the correctness of Algorithm FCS.

Theorem 6.3 *Algorithm FCS returns TRUE if and only if there is a chordal sandwich between G_1 and G_2.*

Proof. If G_1 is a clique correctness follows trivially. In the rest of the proof we will assume that G_1 is not a clique. For a full block (C, S) of G_1 define $Cs(R(C, S))$ to be TRUE if there is a chordal sandwich between $R(C, S)$ and $G_2[C \cup S]$ and FALSE otherwise. Notice that there is a chordal sandwich between G_1 and G_2 if and only if there is a chordal sandwich between G_1 and G_2 that

Algorithm: Find Chordal Sandwich – `FCS` (G_1, G_2)

Input: Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ so that $G_1 \subseteq G_2$, together with a list Π_1 of all potential maximal cliques of G_1 that induce cliques in G_2.

Output: TRUE if there is a chordal sandwich between G_1 and G_2, FALSE otherwise.

$\Delta_1 := \{S \in \Delta(G_1) : \text{There is an } \Omega \in \Pi_1 \text{ so that } S_1 \subset \Omega\}$;

$\mathcal{F}_1 :=$ the set of all full blocks (S, C) so that $S \in \Delta_1$, sorted by $|S \cup C|$;

$Cs(R(S, C)) :=$ FALSE for every pair of vertex sets S and C;

foreach full block (S, C) in $\mathcal{F}_1$ taken in ascending order **do**
    foreach potential maximal clique $\Omega \in \Pi_1$ so that $S \subset \Omega \subseteq S \cup C$ **do**
        $ok :=$ TRUE;
        foreach full block (S_i, C_i) where $C_i \in \mathcal{C}_{G_1}(\Omega)$ and $S_i = N(C_i)$ **do**
            if $Cs(R(S_i, C_i)) =$ FALSE **then**
                $ok :=$ FALSE;
        $Cs(R(S, C)) := Cs(R(S, C)) \vee ok$;

if G_1 is a clique **then**
    RETURN TRUE;

else
    RETURN $\bigvee_{S \in \Delta_1} \bigwedge_{C \in \mathcal{C}_{\mathcal{G}(\mathcal{S})}} Cs(R(S, C))$;

is a minimal triangulation of G_1. It follows directly from this and Lemma 6.1 that there is a chordal sandwich between G_1 and G_2 if and only if G_1 has a minimal separator S so that for every full block (S, C) there is a chordal sandwich between $R(S, C)$ and $G_2[S, C]$. We give a recurrence relation for $Cs(R(C, S))$, and the correctness of algorithm FCS follows directly from this recurrence relation. We wish to show that $Cs(R(C, S)) = TRUE$ if and only if there is a potential maximal clique $\Omega \in \Pi_1$ so that $S \subset \Omega \subseteq S \cup C$ and so that for every full block (S_i, C_i) with $C_i \in \mathcal{C}(\Omega)$ and $S_i = N(C_i)$, $Cs(R(C_i, S_i)) = TRUE$. We prove each direction of the statement above by induction on $|S \cup C|$. If $|S \cup C| = 0$ the statement clearly is true. Suppose now that the statement is true whenever $|S \cup C| < k$ for some $k > 0$ and consider a full block of size k.

Suppose there is a chordal sandwich between $R(C, S)$ and $G_2[C \cup S]$. Then, there is a chordal sandwich G' between $R(C, S)$ and $G_2[C \cup S]$ that is a minimal triangulation of $R(C, S)$. Let Ω be a maximal clique of G' that contains S. Thus, Ω is a potential maximal clique of $R(C, S)$. It follows directly from Theorem 6.2 that every potential maximal clique of a realization of a full block of G_1 is a potential maximal clique of G_1, thus we know that Ω is a potential maximal clique of G_1. As $G' \subseteq G_2$, Ω induces a clique in G_2 so Ω is a potential maximal clique in Π_1 satisfying $S \subset \Omega \subseteq S \cup C$. Let (S_i, C_i) be a full block of G_1 with $C_i \in \mathcal{C}_{G_1}(\Omega)$ if one such exists, if not, the statement follows directly. Now $G'[S_i \cup C_i]$ is clearly a chordal sandwich between $R(S_i, C_i)$ and $G_2[S_i \cup C_i]$ and $Cs(R(S_i, C_i)) = TRUE$ by the induction hypothesis.

In the other direction, suppose there is a potential maximal clique $\Omega \in \Pi_1$ so that $S \subset \Omega \subseteq S \cup C$ and so that for every full block (S_i, C_i) with $C_i \in \mathcal{C}_{G_1}(\Omega)$ and $S_i = N(C_i)$, $Cs(R(C, S)) = TRUE$. By the induction hypothesis, there is a chordal sandwich G'_i between $R(S_i, C_i)$ and $G_2[S_i \cup C_i]$ for every full block (S_i, C_i) with $C_i \in \mathcal{C}_{G_1}(\Omega)$ and $S_i = N(C_i)$. Observe that for any $i \neq j$ $(S_i \cup C_i) \cap (S_j \cup C_j) \subseteq \Omega$. Thus we can build a graph G' so that $G'[\Omega]$ is a clique and $G'[S_i \cup C_i] = G'_i$. Clearly, G' is chordal supergraph of $R(S, C)$. As $\Omega \in \Pi_1$ we know that $G_2[\Omega]$ is a clique, and that G'_i is a subgraph of $G_2[S_i \cup C_i]$. Thus G' is a chordal sandwich between $R(S, C)$ and $G_2[S \cup C]$. ∎

Theorem 6.4 *Algorithm FCS terminates in $O^*(|\Pi_1|)$ time.*

Proof. Computing Δ_1 from Π_1 can be done in $O^*(|\Pi_1|)$ time by looping over each potential maximal clique $\Omega \in \Pi_1$ and inserting $N(C)$ into Δ_1 unless already present for every connected component C of $G \backslash \Omega$. $\mathcal{F}_1$ can be computed similarly and then sorted in $O^*(|\Pi_1|)$ time. While building Δ_1 and $\mathcal{F}_1$ we can store a pointer from every full block $(S, C) \in \mathcal{F}$ to all potential maximal cliques $\Omega \in \Pi_1$ satisfying $S \subset \Omega \subseteq S \cup C$. Using these pointers, in each iteration of the second **foreach** loop we can find the next potential maximal clique Ω to consider in constant time. Furthermore, it is easy to see that each iteration of the second **foreach** loop runs in polynomial time. Thus, the total running time is bounded by $O^*(\sum_{(S,C)\in\mathcal{F}_1} |\{\Omega \in \Pi_1 : S \subset \Omega \subseteq S \cup C\}|) = O^*(\sum_{\Omega\in\Pi_1} |\{(S, C) \in \mathcal{F}_1 : S \subset \Omega \subseteq S \cup C\}|)$. But as $|\{(S, C) \in \mathcal{F}_1 : S \subset \Omega \subseteq S \cup C\}| \leq n$ for

every potential maximal clique Ω, it follows that the algorithm runs in time $O^*(\sum_{\Omega \in \Pi_1} |\{(S,C) \in \mathcal{F}_1 : S \subset \Omega \subseteq S \cup C\}|) = O^*(|\Pi_1|)$. ∎

Theorem 6.5 *[9] $\Pi(G)$ can be listed in $O^*(1.7549^n)$ time. Thus $|\Pi(G)| = O^*(1.7549^n)$.*

Corollary 6.6 *There is an algorithm that solves the Chordal Sandwich problem in time $O^*(1.7549^n)$.*

Proof. Compute $\Pi(G)$. By Theorem 6.5 this can be done in $O^*(1.7549^n)$ time. Now, for every $\Omega \in \Pi(G)$ we can test in $O(n^2)$ time whether it is a clique in G_2. If it is, insert Ω into Π_1. We can now call algorithm FCS on G_1, G_2 and Π_1, and return the same answer as algorithm FCS. By Theorem 6.4 algorithm FCS terminates in time $O^*(|\Pi_1|) = O^*(1.7549^n)$ completing the proof. ∎

Corollary 6.7 *There is an algorithm that solves the Chordal Sandwich problem in time $O^*(2^{tw(G_2)})$ where $tw(G_2)$ is the treewidth of G_2.*

Proof. For any tree-decomposition of G_2, every clique of G_2 is contained in some bag in this tree-decomposition [13]. Thus, G_2 has at most $O^*(2^{tw(G_2)})$ cliques. We can list all cliques of a graph with a polynomial delay [15]. For every clique Ω in G_2 we can test whether it is a potential maximal clique of G_1 in polynomial time [2]. If it is, we insert Ω into Π_1. Thus $|\Pi_1| = O^*(2^{tw(G_2)})$. Finally, call algorithm FCS on G_1, G_2 and Π_1, and return the same answer as algorithm FCS. By Theorem 6.4 algorithm FCS terminates in time $O^*(|\Pi_1|) = O^*(2^{tw(G_2)})$ completing the proof. ∎

Corollary 6.8 *There is an algorithm for Weighted k-Treelength that runs in time $O^*(1.7549^n)$.*

Proof. By Observation 3.1 $tl_w(G) \leq k$ if and only if there is a chordal sandwich between G and G_w^k. By Corollary 6.6 we can check this in time $O^*(1.7549^n)$. ∎

7 Conclusion

We have proved that it is NP-complete to recognize graphs with treelength bounded by a constant $k \geq 2$. In addition we have proved that unless $P = NP$ there can be no approximation algorithm for the treelength of weighted graphs with approximation factor better than $\frac{3}{2}$ and we believe that a similar result holds for unweighted graphs as well. Finally we gave a $O^*(1.7549^n)$ algorithm to solve the Chordal Sandwich problem and showed how it can be used to determine the treelength of a graph within the same time bound. Dourisboure and Gavoille provide two 3-approximation algorithms for treelength in [6], and propose a heuristic that they conjecture is a 2-approximation algorithm. It would be interesting to see whether the gap between the upper and lower bounds for approximability of treelength can be closed.

References

[1] H.L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[2] V. Bouchitte and I. Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM Journal on Computing*, 31(1):212–232, 2001.

[3] P. Buneman. A characterization of rigid circuit graphs. *Discrete Mathematics*, 9:205–212, 1974.

[4] Y. Dourisboure. Compact routing schemes for bounded tree-length graphs and for k-chordal graphs. *Proceedings DISC 2004, Lecture Notes in Computer Science*, 3274:365–378, 2004.

[5] Yon Dourisboure, Feodor F. Dragan, Cyril Gavoille, and Chenyu Yan. Spanners for bounded tree-length graphs. *Theor. Comput. Sci.*, 383(1), 2007.

[6] Yon Dourisboure and Cyril Gavoille. Tree-decompositions with bags of small diameter. *Discrete Mathematics*, 307(16):2008–2029, 2007.

[7] U. Feige, M.T. Hajiaghayi, and J.R. Lee. Improved approximation algorithms for minimum-weight vertex separators. In 37^{th} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 563–572. ACM Press, May 2005.

[8] Fedor V. Fomin, Dieter Kratsch, Ioan Todinca, and Yngve Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM J. Comput.*, 38(3), 2008.

[9] Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. In *ICALP (1)*, pages 210–221, 2008.

[10] D.R. Fulkerson and O.A Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965.

[11] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graps. *Journal of Combinatorial Theory B*, 16:47–56, 1974.

[12] M.C. Golumbic, H. Kaplan, and R. Shamir. Graph sandwich problems. *J. Algorithms*, 19(3):449–473, 1995.

[13] P. Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

[14] T. Kloks, D. Kratsch, and J. Spinrad. On treewidth and minimum fill-in of asteroidal triple-free graphs. pages 309–335, 1997.

[15] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. pages 260–272, 2004.

[16] J. Walter. *Representation of rigid cycle graphs.* PhD thesis, 1972.