THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENTS OF COMPUTER SCIENCE AND ENGINEERING AND MATHEMATICS

THE NP-COMPLETENESS OF FINDING TREEWIDTH FOR GRAPHS AND A 4-APPROXIMATION FOR FINDING TREEWIDTH

DANIEL HOFMAN
SPRING 2019

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Computer Engineering and Mathematics
with interdisciplinary honors in Computer Engineering and Mathematics

Reviewed and approved* by the following:

Martin Fürer
Professor of Computer Science
Thesis Supervisor

Chita Das
Professor of Computer Science and Engineering
Department Head
Honors Adviser

Sergei Tabachnikov
Professor of Mathematics
Honors Adviser

*Signatures are on file in the Schreyer Honors College.

Abstract

Knowing certain parameters related to a graph can make solving problems known to be hard tractable when the parameter is fixed. One such widely studied parameter is treewidth, or a graph's closeness to a tree. Therefore, finding the treewidth of a graph can be very useful. Here, we present findings on the NP-completeness of finding treewidth for arbitrary graphs as well as a useful 4-approximation algorithm for finding the treewidth of a graph.

Table of Contents

Chapter 1

Treewidth, Branchwidth, and Clique-width

1.1 Motivation

There are many classic problems in graph theory which have been proven to be NP-hard, such as Vertex Cover, Vertex Coloring, and Hamiltonian Path. Although these problems are known to be hard for arbitrary graphs, they may still be easy for special cases of graphs. Therefore, there is great reason to find out what these cases might be because in practice most graphs may fall into these special cases.

One special case that is natural to consider is tree graphs. We can find a minimal vertex cover for a tree in polynomial time by giving the tree a root and recursively deciding whether a root or its children are in the cover. Similarly, any tree is colorable with two colors.

Since many problems are easier for trees, they are also easier for graphs that are close to a tree, such as a graph with only one cycle. Therefore, it is natural to consider ways to measure how close a graph is to being a tree. We call this measurement treewidth.

Using treewidth, we can use techniques that solve hard problems for trees and extend them to graphs that are close to trees. In fact, as we will show, if we know that a graph has treewidth k, we can solve some hard problems for the graph in time polynomial in the size of the graph, but exponential in k. Hence, graphs with small treewidth may have practical solutions to NP-hard problems.

1.2 Defining Treewidth

Treewidth is a parameter that measures a graph's closeness to a tree. To define treewidth, we first define a tree decomposition of a graph. The tree decomposition of a graph G is a tree that has information about G.

Definition 1.2.1 (Tree decomposition [1])**.** A tree decomposition of a graph $G = (V, E)$ is a tree $\mathcal{T}$ together with a collection of subsets T_x (called bags) of V labeled by the vertices x of $\mathcal{T}$ such that $\cup_{x \in \mathcal{T}} T_x = V$ and the following connectivity properties (1) and (2) hold:

(1) For every edge $(u, v) \in E$, there is some x such that $\{u, v\} \subset T_x$.

(2) If y is a vertex on the unique path in $\mathcal{T}$ from x to z, then $T_x \cap T_z \subset T_y$.

The tree decomposition of a graph is itself a tree. This tree is usually involved in algorithms based on treewidth since a solution to a problem for a graph's tree decomposition can extend to the graph itself with some extra work.

Now we define treewidth.

Definition 1.2.2 (Tree decomposition width [1])**.** The width of a tree decomposition $\mathcal{T}$ is the maximum value of $|T_x| - 1$ taken over all vertices x of $\mathcal{T}$.

Definition 1.2.3 (Treewidth [1])**.** The treewidth of a graph G is the minimum k such that G has a tree decomposition with width k.

It is useful to prove some properties about tree decompositions to highlight some of their characteristics.

Lemma 1.2.1. *Let $\{T_x \mid x \in \mathcal{T}\}$ be a tree decomposition of $G = (V, E)$.*
(i) *Let $x \in V$. Then the collection of $y \in \mathcal{T}$ with $x \in T_y$ forms a subtree of $\mathcal{T}$.*
(ii) *Suppose that C is a clique of G. Then there is some $x \in \mathcal{T}$ with $C \subset T_x$.*

Proof. (i) Let $Y = \{y \in \mathcal{T} \mid x \in T_y\}$ be this collection. Since $\mathcal{T}$ is a tree and Y is a subset of this tree, Y is itself acyclic. It remainds to show that Y is connected.

Suppose there are $w, z \in Y$ such that there is no path from w to z in Y. Since $\mathcal{T}$ is a tree, there is a unique path in $\mathcal{T}$ from w to z, and there must be some vertex v on this path such that $v \notin Y$. If no such v existed, there would be a path from w to z in Y.

Using the second property of tree decompositions, $T_w \cap T_z \subset T_v$. By the definition of Y, $x \in T_w$ and $x \in T_z$. However, since $v \notin Y$, $x \notin T_v$, a contradiction. Hence, Y must be connected.

(ii) We prove this by induction on the size of the clique C. For our base case, suppose C is a clique of size 2, i.e. two connected vertices. Then the claim is trivially satisfied by the first property of tree decompositions.

Now suppose the claim holds for cliques of size k. Let C be a clique of size $k + 1$. Suppose there is a tree decomposition $\mathcal{T}$ of G such that C is not a subset of any of the bags. Since $k+1 \geq 3$, there are at least 3 distinct cliques of size k that are a subset of C. By our induction hypothesis they must each be contained in a bag of $\mathcal{T}$. Furthermore, these bags must be distinct, for if any of them coincided they would have to contain all of C, contradicting our assumption.

Let $a, b, c \in V$, be the three unique vertices excluded in the three distinct subset cliques of C. Each of these vertices is included in the two other bags. Therefore, these two other bags must have an edge between them since, as proven above, the collection of bags containing this vertex forms a subtree. Therefore, all three bags must be pairwise connected, forming a cycle. This contradicts that $\mathcal{T}$ is a tree. Therefore, any tree decomposition of G must have a bag containing C. □

It is easy to construct a tree decomposition of a tree with width 1. Every bag contains just one edge from the tree. In addition, using the properties proven above we can easily show that a graph with a cycle must have treewidth at least 2. Therefore, a graph has treewidth 1 if and only if it is a tree. The closer a graph's treewidth is to 1, the closer it is to being a tree.

Chapter 2

NP-Completeness of Finding Treewidth

2.1 Partial k-trees

Before delving into a proof that finding the treewidth of a graph is NP-complete, we first formulate an equivalent definition of the treewidth of a graph using k-trees.

Definition 2.1.1 (k-tree)**.** The class of k-trees is the smallest class obeying properties (i) and (ii) below.

(i) K_{k+1}, the complete graph on $k+1$ vertices, is a k-tree.

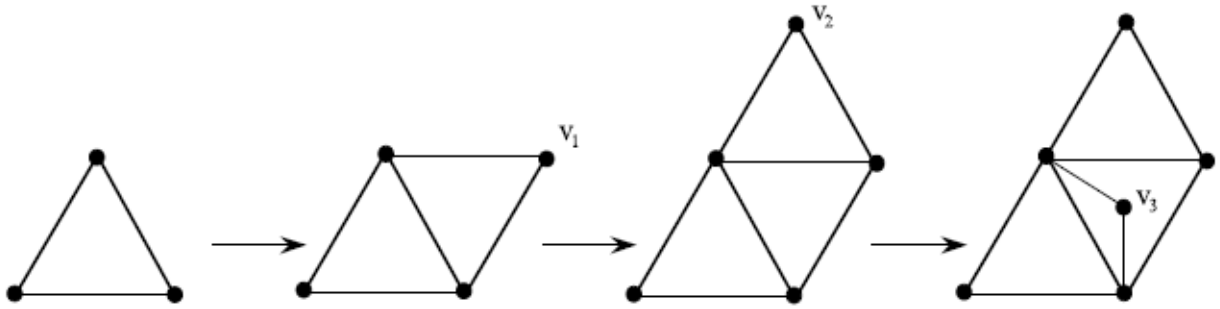
(ii) If G is a k-tree and H is a subgraph of G isomorphic to K_k, then the graph G' constructed from G by first adding a new vertex v to G and then adding edges to make $H \cup \{v\}$ a copy of K_{k+1}, is a k-tree.

Definition 2.1.2 (Partial k-tree)**.** If G is a subgraph of a k-tree, then G is called a partial k-tree.

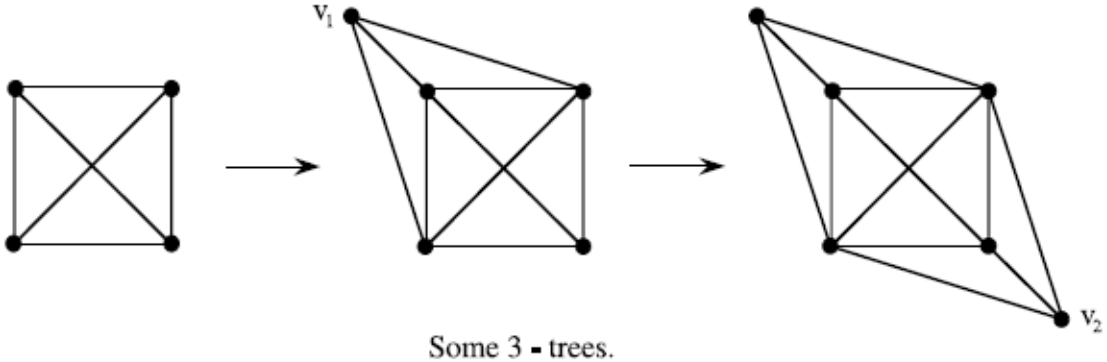
Figure 2.1 shows some examples of 2-trees and 3-trees and also demonstrates how they can be built up using the iterative definition.

From the definition of k-trees, one can see how a k-tree has treewidth k. Each of the cliques of size $k+1$ in a k-tree gets its own bag in a tree decomposition, and the bags are connected by edges in the same manner as the cliques are connected by adding an extra vertex at each step.

We formalize the connection between treewidth and partial k-trees below.



Some 2 - trees.



Some 3 - trees.

Figure 2.1: Examples of some 2-trees and 3-trees. Figure from [1].

Theorem 2.1.1 ([1]). *A graph G has a tree decomposition of width k if and only if G is a partial k-tree.*

Proof. ($\Rightarrow$) Here, we prove the stronger claim that if a graph G has a tree decomposition $\{T_x : x \in \mathcal{T}\}$ of width k, then G is a partial k tree, and G is a subgraph of a k-tree $K(G)$ in which every subset of a bag with $\leq k$ elements is part of a k-clique. Suppose this were not the case. Then there is some minimal graph G such that the above does not hold.

This graph G is not empty since the above holds for complete graphs K_{k+1}. They have a trivial tree decomposition of width k, and $K(G) = G$.

Let x be a leaf of $\mathcal{T}$, G's tree decomposition. Let $\mathcal{T}'$ be the result of removing x from $\mathcal{T}$ and G' a maximal subgraph of G corresponding to $\mathcal{T}'$. Note that $\mathcal{T}'$ is still a tree decomposition of width k since $\max_{x \in \mathcal{T}'} |T_x|$ does not increase. Let y be the neighbor of x in $\mathcal{T}$. Let $T_x \cap T_y = T'_x$ and $T_x - T'_x = T''_x$. If $T''_x = \emptyset$, then $T_x \subset T_y$ and $G = G'$, so we can repeat this process. Hence, suppose that $T''_x \neq \emptyset$. In this case G' is smaller than G, so by our hypothesis G' is a subset of a k-tree $K(G')$ with the property above.

T'_x is a proper subset of T_x, so it has at most k elements since the width of $\mathcal{T}$ is k. Hence, T'_x is at most k elements in T_y, a bag of $\mathcal{T}'$, so by our hypothesis, T'_x is part of a k-clique C in $K(G')$. Now we add vertices from T''_x to $K(G')$ one vertex at a time using the following algorithm.

1. Let $K = K(G')$
2. Pick a vertex v of T''_x.
3. Add v to K and add all edges between vertices of C and v.
4. Let $T''_x \leftarrow T''_x - \{v\}$
5. If there is any vertex $c \in C - T_x$, let $C \leftarrow (C - \{c\}) \cup \{v\}$.
6. If $T_x \neq \emptyset$ goto step 2.

Because C has k elements, T''_x has at most k elements, and each loop of the algorithm adds one element of T''_x while removing one original element from the clique, C is a k-clique of K after each iteration. Hence, when the algorithm terminates, K is still a k-tree since vertices are added exactly as in the definition of k-trees. Moreover, T'_x begins as a clique itself since it is a subgraph of a k-clique and each time a vertex from T''_x is added, edges are added connecting it to each vertex of T'_x and eah vertex of T''_x already added. Hence, the vertices of T_x form a clique in the k-tree K. Therefore, G is a subgraph of K since G' was only missing some vertices of T_x and edges between these vertices in G and these vertices and all such possible edges between them have been added. Hence $K(G) = K$ with the desired properties, contradicting that G is the minimal graph that this is not the case for.

($\Leftarrow$) First we prove that if $G = (V, E)$ is a k-tree, then it has a tree decomposition of width k such that each k-clique of G occurs in some bag of its tree decomposition. We prove this by induction on $|V|$, the number of vertices of G.

For the base case, consider K_{k+1}. Clearly, this graph has the trivial tree decomposition where all vertices are in one bag.

Now consider the case that $|V| > k + 1$. Assume our induction hypothesis for all k-trees with less than $|V|$ vertices. Let v be the last vertex added in the construction of G. By the definition of k-trees, the neighbors of v form a k-clique C. Let $G' = G - \{v\}$. G' is a k-tree with $|V| - 1$

vertices, so it has a tree decomposition $\mathcal{T}'$ of width k with the property above. Since C is a k-clique, it occurs in some bag T_x for $x \in \mathcal{T}'$. Now we construct $\mathcal{T}$ by adding a leaf y to $\mathcal{T}'$ adjacent to x with $T_y = C \cup \{v\}$. Since $\mathcal{T}'$ has width k and $|T_y| = K + 1$, $\mathcal{T}$ also has width k. This tree decomposition also corresponds to G since G' was only missing v and all of the edges from C to v in G, and they are now included in $\mathcal{T}$. The only k-cliques added to G' are also all included in the bag T_y. Hence, our induction hypothesis also holds for $|V|$.

Now suppose G is a partial k-tree. Then, there is some supergraph K of G such that K is a k-tree. K has a tree decomposition of width k, and this same tree decomposition is also a tree decomposition for G. □

Corollary 2.1.1.1. *A graph G has treewidth k if and only if k is the least integer such that G is a partial k-tree.*

Proof. ($\Rightarrow$) Suppose G has treewidth k. Then it has a tree decomposition of width k, so it is a partial k-tree. If G were also a partial h-tree for some $h < k$, then it would have a tree decomposition of width h, contradicting that the treewidth of G is k. Hence, k is the least integer such that G is a partial k-tree.

($\Leftarrow$) Suppose that k is the least integer such that G is a partial k-tree. Since G is a partial k-tree, it has a tree decomposition of width k. Suppose it had a tree decomposition of with h for some $h < k$. Then G is also a partial h-tree, contradicting the hypothesis. □

We have now established an equivalent definition for treewidth. Therefore, we can prove that finding the treewidth of a graph is NP-complete by proving that finding the least k such that G is a k-tree is NP-complete.

We define the problem PARTIAL K-TREE as follows: Given a graph G and a positive integer k, is G a partial k-tree? If we can solve this decision problem in polynomial time, then we can find the treewidth of a graph in polynomial time by checking whether a graph is a partial k-tree for various values of k.

In the following sections, we will show that the problem of finding the treewidth of a graph is NP-complete by proving that PARTIAL K-TREE is NP-complete.

2.2 Chordal Graphs and Elimination Schemes

Now, we explore another characterization of k-trees: chordal graphs. We will work through some definitions and lemmas critical to the NP-completeness proof.

Definition 2.2.1 (Chordal graph)**.** A graph is chordal if every cycle of length greater than length three has a chord (an edge between two nonadjacent vertices in the cycle).

We will now explore an equivalent definition of a chordal graph involving elimination schemes.

Definition 2.2.2 (Elimination scheme)**.** An elimination scheme of a graph is an ordering π of its vertices.

Definition 2.2.3 (Fill edge)**.** An edge (u, v) in a graph G is a fill edge with respect to the elimination scheme π if there is a vertex v preceding u and w in π such that both u and w are adjacent to v via original or fill edges but not to each other.

Definition 2.2.4 (Filled graph)**.** The filled graph of $G = (V, E)$ with respect to π is the graph $G' = (V, E \cup F^{\pi})$ where F^{π} is the set of fill edges of G with respect to π.

Definition 2.2.5 (Perfect elimination scheme)**.** An elimination scheme of G is perfect if it has no fill edges.

Even though an elimination scheme is just a permutation of the vertices of a graph, we can think of it as an order we use to remove vertices. Each time we remove a vertex from the graph, we can add the fill edges the vertex induces. In a perfect elimination scheme, we never have to add any fill edges each time we remove a vertex. This fact motivates an equivalent definition of a perfect elimination scheme involving simplicial vertices.

Definition 2.2.6 (Simplicial vertex)**.** A vertex v is simplicial if its neighbors in G form a clique.

Lemma 2.2.1. *An elimination scheme π of G is a perfect elimination scheme if and only if each vertex of G is simplicial at the time of its elimination.*

Proof. ($\Rightarrow$) Suppose a vertex v is not simplicial at the time of its removal in π. Then v has two neighbors u, w which are not adjacent to each other. Therefore, (u, w) is a fill edge, so π is not perfect.

($\Leftarrow$) Suppose every vertex is simplicial at the time of its elimination from G under π but the elimination scheme is not perfect. Let (u, w) be the first fill edge. Then there must be no edge between u and w and a vertex v preceding u and w in π and adjacent to both u and w. This fact contradicts that v is simplicial at the time of its removal. □

Definition 2.2.7 (Block)**.** A block of a graph G is a maximal set of vertices with the same closed neighborhood. A block-contiguous elimination scheme is one in which entire blocks are eliminated at a time.

Lemma 2.2.2. *A graph G has a perfect elimination scheme if and only if it has a block-contiguous perfect elimination scheme.*

Proof. ($\Leftarrow$) Trivial.

($\Rightarrow$) Let π be a perfect elimination scheme for G. Let π' be an elimination scheme which eliminates blocks in the order that they first appear in π. In other words when a vertex v appears in π, then it appears in π' and then every other vertex in the block containing v is eliminated in any order. Then the order continues with the next vertex in π not eliminated yet.

I claim that π' is a perfect elimination scheme for G. To see this, we first consider that if any vertex in a block is simplicial, all vertices in the block are simplicial. This is because if v and u are in the same block, and v is simplicial, then u is a part of the clique that is the neighborhood of v. But the neighborhood of u is only these vertices and v since u and v are in the same block, and v is neighbors of each of these vertices, so u is also simplicial. Next, we consider that if we remove a simplicial vertex from a block, then the remaining vertices in the block are still simplicial. This is because each of their neighborhoods were cliques before, and those cliques each just lost vertex v, so they remain cliques. Hence, when we eliminate the first vertex from a block in π', we can eliminate the rest of the vertices in the block in any order. Additionally, the next block selected was the earliest remaining vertex in the order of π. Since this vertex was simplicial in the order of π, it must also be simplicial in the order of π' since its neighbors are a subset of those in π. □

Now we have the tools to connect chrodal graphs to elimination schemes.

Lemma 2.2.3. *A graph is chordal if and only if it has a perfect elimination scheme.*

Proof. ($\Leftarrow$) Suppose a graph G has a perfect elimination scheme π. Let C be a cycle in G with at least four vertices. Let $v \in C$ be the first vertex of the cycle in the order π. Then v is adjacent to the two vertices next to it in the cycle, and these two vertices appear later in the ordering π. Hence, there must be a chord between them since, by Lemma 2.2.1, v is simplicial at the time of its elimination.

($\Rightarrow$) Proven in [2]. □

The connection between k-trees and chordal graphs may be clearer now. Clearly, any k-tree has a perfect elimination scheme. By removing vertices in reverse order of how they are added in the inductive definition of a k-tree, each vertex will be simplicial at the time of its removal. Hence, all k-trees are chordal graphs.

Definition 2.2.8 (k-chordal graph)**.** A k-chordal graph is a chordal graph in which the maximum clique size is $k + 1$. A partial k-chordal graph is a subgraph of a k-chordal graph.

Definition 2.2.9. Given a graph G, we define $k_t(G)$ to be the minimum k such that G is a partial k-tree. We define $k_c(G)$ to be the minimum k such that G is a partial k-chordal graph.

From the definition of k-chordal graphs, in every perfect eleimination scheme of a k-chordal graph, the neighborhood of any vertex at the time of its elimination forms a clique of size at most k. Clearly, any k-tree is a k-chordal graph, and any k-chordal graph is a partial k-tree. We now have the tools to connect partial k-chordal graphs with partial k-trees.

Lemma 2.2.4. *For any graph G, $k_t(G) = k_c(G)$.*

Proof. Since k-chordal graphs are partial k-trees, we have by definition that $k_c(G) \leq k_t(G)$. To show that $k_t(G) \leq k_c(G)$, we refer to [3]. □

We next need to define chain graphs for use in the NP-completeness proof.

Definition 2.2.10 (Chain graph)**.** A bipartite graph $G = (A \cup B, E)$ is a chain graph if there exists a permutation τ of the vertices of A such that for any two vertices $u, v \in V$, we have $\tau(u) < \tau(v)$ if and only if $\Gamma(u) \subset \Gamma(v)$, where $\Gamma(w)$ for a vertex w is the set of the neighbors of w in G.

Definition 2.2.11. For a given bipartite graph $G = (A \cup B, E)$, the graph $C(G)$ is formed from G by adding edges to make A and B complete subgraphs.

The following lemma is proven in [4].

Lemma 2.2.5. *A bipartite graph $G = (A \cup B, E)$ is a chain graph if $C(G)$ is chordal.*

Lemma 2.2.6. *If a bipartite graph $G = (A \cup B, E)$ is a chain graph then $C(G)$ has a perfect elimination scheme starting with all vertices in A.*

Proof. Since G is a bipartite graph, it has a chain bijection τ of A. Let π be a reverse ordering of τ followed by any ordering of B. We claim that this is a perfect elimination scheme for $C(G)$. Suppose this was not the case. Then $C(G)$ has a fill edge with respect to π. This edge cannot be an edge in the set A or B since those sets form cliques. Hence, the fill edge must be an edge with one vertex in A and one vertex in B. Let these vertices be a and b respectively. Then there is a vertex v coming earlier in the order π than a or b such that v is adjacent to a and b while a and b are not adjacent. Since v is earlier in the order π, it is later in the order τ. Hence, $\Gamma(a) \subset \Gamma(v)$. But we know that b is a neighbor of v, so b must be a neighbor of a, contradicting that (a, b) is a fill edge. □

2.3 NP-Completeness

To prove that PARTIAL K-TREE is NP-complete, we need a reduction from another problem known to be NP-complete. We use MINIMUM CUT LINEAR ARRANGEMENT (MCLA).

Definition 2.3.1. Given a graph $G = (V, E)$ and a permutation τ of V, we define the linear cut value of G with respect to τ as

$$c_\tau(G) = \max_{1 \le i < |V|} \left| \{(u, v) \in E : \tau(u) \le i < \tau(v)\} \right|.$$

MCLA is defined as follows. Given a graph G and a positive integer k, does there exist a permutation τ of V such that $c_\tau(G) \le k$? MCLA is known to be NP-complete. See [5] for more info on MCLA.

In constructing a reduction from MCLA to PARTIAL K-TREE, when given graph $G = (V, E)$, we create a bipartite graph $G' = (A \cup B, E')$. In the following descriptions, we use $\Delta(G)$ to mean the maximum degree of a vertex in G. For each vertex $v \in V$, we create a set A_v of $\Delta(G) + 1$ vertices in A and a set B_v of $\Delta(G) - \deg_G(v) + 1$ vertices in B. For each edge $e \in E$, we create a set B_e of two vertices in B. For each $v \in V$, every vertex in A_v is adjacent to every vertex in B_v. Additionally, for each $v \in V$ and each $e \in E$, every vertex in A_v is adjacent to every vertex in B_e if v is an endpoint of e.

An example of this construction can be seen in Figure 2.2. It is easy to see that the A_v, B_v, and B_e sets make up the blocks of the graph $C(G')$.

We now begin the NP-completeness proof by demonstrating how the reduction connects linear cut values with k-trees.

Lemma 2.3.1 ([3])**.** *Given a graph G and a positive integer k, G has a minimum linear cut value k w.r.t. some permutation π if and only if the corresponding graph $C(G')$ is a partial k'-tree for $k' = (\Delta(G) + 1)(|V| + 1) + k - 1$.*

Proof. ($\Leftarrow$) Suppose that $C(G')$ is a partial k'-tree. Then by Lemma 2.2.4, $C(G')$ is a partial k'-chordal graph. Let F be a k'-chordal supergraph of $C(G')$. By Lemma 2.2.5, since F is chordal, its edges between A and B form a chain graph. Then by Lemma 2.2.6, F has a perfect elimination scheme π' starting with all vertices in A. Using the process in Lemma 2.2.2, we can also assume without loss of generality that π' is contiguous in the A_v blocks. Since F is k'-chordal, it has no cliques with more than $k' + 1$ vertices, so no vertex has degree greater than k' when it is eliminated from F under π'.

Let π be the ordering of blocks in A induced by π'. Assume without loss of generality, that the vertices of G are numbered in order π and that the vertices are identified by their numbers.

Consider the graph F_i resulting from elimination of vertices of the first $i-1$ blocks of F. Let v be a vertex in A_i. v is adjacent to the $\Delta(G)$ other vertices of A_i. v is also adjacent to all $\Delta(G)+1$ vertices of $A_{i+1}, \ldots, A_{|V|}$ since A forms a clique in $C(G')$. v must also be adjacent to the $\Delta(G)+1-\deg_G(j)$ vertices of B_j for $j=1,\ldots,i$. This is because for $j<i$, when A_j was eliminated, its vertices were adjacent to v and those of B_j, so there must be edges from v to the vertices of B_j for the elimination scheme to be perfect. Finally, v is also adjacent to the two vertices of B_e for each edge e incident to at least one vertex in $\{1,\ldots,i\}$ since there similarly must be those fill edges for the elimination scheme to be perfect. Summing these adjacencies we have that

$$\deg_{F_i}(v) \geq \Delta(G) + (\Delta(G)+1)(|V|-i) + (\Delta(G)+1)i - \sum_{j=1}^{i} \deg_G(j) + 2\left|E_1^i\right| + 2\left|E_2^i\right|,$$

where E_1^i is the set of edges of G with exactly one vertex in $\{1,\ldots,i\}$ and E_2^i is the set of edges of G with both vertices in $\{1,\ldots,i\}$. We have that $\sum_{j=1}^{i}\deg_G(j) = |E_1^i| + 2|E_2^i|$ since each edge fully contained in $\{1,\ldots,i\}$ would add 2 to the sum and each edge with only one vertex in

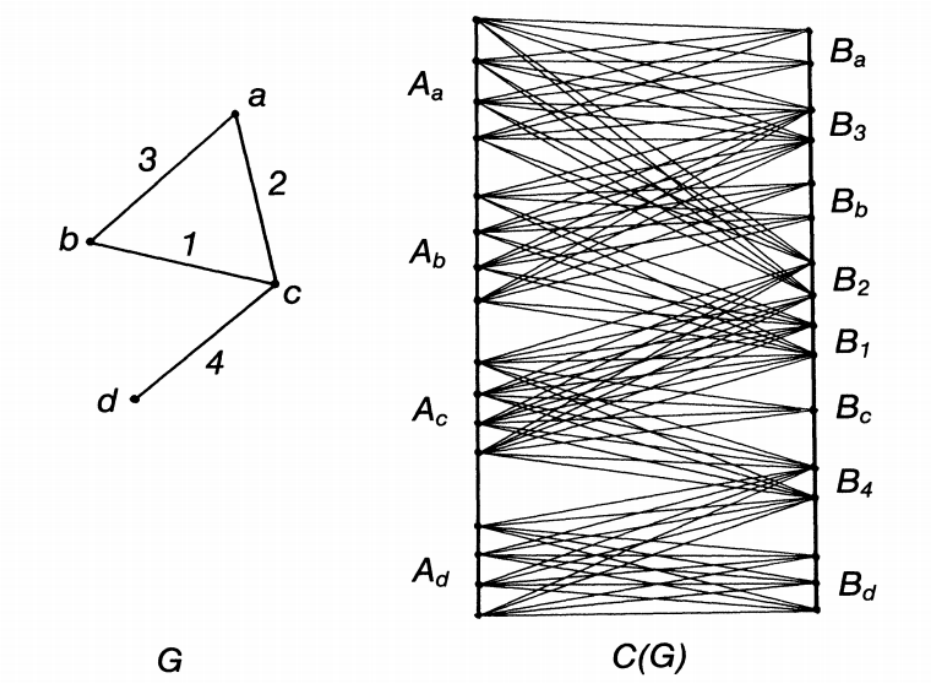


Figure 2.2: Example of $C(G')$. Figure from [3].

$\{1, \ldots, i\}$ would add 1 to the sum. Hence, we have

$$\begin{aligned}
\deg_{F_i}(v) &\geq \Delta(G) + (\Delta(G)+1)(|V|-i) + (\Delta(G)+1)i - \sum_{j=1}^{i} \deg_G(j) + 2\left|E_1^i\right| + 2\left|E_2^i\right| \\
&= \Delta(G) + (\Delta(G)+1)(|V|-i) + (\Delta(G)+1)i + \left|E_1^i\right| \\
&= \Delta(G) + (\Delta(G)+1)|V| + \left|E_1^i\right| \\
&= (\Delta(G)+1)(|V|+1) + \left|E_1^i\right| - 1
\end{aligned}$$

Since at each stage F_i of the elimination scheme, the next vertex, a vertex in A_i, must have less than k' neighbors, we have that for each $i \in \{1, \ldots, |V|-1\}$, $(\Delta(G)+1)(|V|+1) + |E_1^i| - 1 \leq \deg_{F_i}(v) \leq k' = (\Delta(G)+1)(|V|+1) + k - 1$. Hence, $|E_1^i| \leq k$ for each $1 \leq i < |V|$. By the definition of E_1^i, this implies that the linear cut value of G with respect to π is at most k, as desired.

($\Rightarrow$) Now suppose that G has minimum cut value k with respect to ordering π. Let π' be an ordering of the vertices of $C(G')$ beginning with the blocks A_v in the order of π, with the vertices in each block taken in any order, and ending with the clique B in any order. Let F be the filled graph of $C(G')$ with respect to π'. Using the same analysis as above, we can find every fill edge and calculate that the degree of each vertex of A_i in graph F_i is $(\Delta(G)+1)(|V|+1) + |E_1^i| - 1$. Since k is the minimum linear cut value of G with respect to π, we have that $|E_1^i| \leq k$ for each $1 \leq i < |V|$. Therefore, the degree of any vertex when it is eliminated in F under π' is at most $(\Delta(G)+1)(|V|+1) + k - 1 = k'$. Hence, F cannot have any cliques of size $k'+1$ or greater, and F is a k'-chordal graph. Since F is a supergraph of $C(G')$, $C(G')$ is a partial k'-chordal graph and, equivalently, a partial k'-tree.

□

Proving that PARTIAL K-TREE is NP-complete follows easily from this Lemma.

Theorem 2.3.2. *PARTIAL K-TREE is NP-complete.*

Proof. Clearly, PARTIAL K-TREE is in NP since it is equivalent to deciding whether a graph is partial k-chordal. Elimination schemes can be used a certificate to test whether a graph is k-chordal in polynomial time by eliminating the vertices according to the elimination scheme and checking whether the number of neighbors of each vertex is at most k at each elimination.

To show that PARTIAL K-TREE is NP-hard, we need that MCLA $\leq_P$ PARTIAL K-TREE. This fact follows from Lemma 2.3.1 and the fact that creating graph $C(G')$ from arbitrary graph G takes polynomial time. □

Chapter 3

Approximating Treewidth

3.1 Separators

Because finding treewdith is NP-complete, it is desirable to find efficient ways to approximate treewidth. Here we present a 4-approximation of treewidth that takes advantage of separators, or vertex sets that disconnect graphs.

Definition 3.1.1. Let $G = (V, E)$ be a graph of treewidth at most k and $W \subset V$. A balanced W-separator is a set $S \subset V$ such that every connected component C of $G \setminus S$ contains at most $\frac{|W|}{2}$ elements of W.

Theorem 3.1.1 ([6]). *Let $G = (V, E)$ be a graph and $k \geq 1$.*

(1) If G has a tree width at most k, then for every $W \subset V$ there exists a balanced W-separator of cardinality at most $k + 1$.

(2) If for every $W \subset V$ with $|W| = 2k + 3$ there exists a balanced W-separator of cardinality at most $k + 1$, then G has a treewidth at most $3k + 3$.

Proof. First we prove statement (1). Let $W \subset V$. Let $\{T_x \mid x \in \mathcal{T}\}$ be a tree decomposition of G with width k. Let $\mathcal{T}$ be a rooted tree. We define the height of a vertex of $\mathcal{T}$ to be the height of its subtree. Let $x \in \mathcal{T}$ be a node of minimum height such that the bags of its subtree contain more than $\frac{|W|}{2}$ elements of W. Set $S = T_x$. Let C_0 be the vertices of $\mathcal{T}$ without the subtree rooted at x and without the vertices of S. Since the subtree rooted at x has more than $\frac{|W|}{2}$ vertices of W, C_0 must have less. Let $y_1, \ldots, y_m$ be the children of x in $\mathcal{T}$, and let C_i be the vertices of the subtree rooted at y_i without the vertices of S for $i = 1, \ldots, m$. By the construction of x, each of these sets must have at most $\frac{|W|}{2}$ vertices. Furthermore, $C_0, C_1, \ldots, C_m$ form the connected components of $G \setminus S$ since every path from a vertex in one of these sets to another would have a vertex in the T_x bag and hence pass through S.

To prove statement (2), we prove the stronger statement that if for every $W \subset V$ with $|W| = 2k + 3$ there exists a balanced W-separator of cardinality at most $k + 1$, then for every $W \subset V$ with $|W| \leq 2k + 3$ the graph G has a rooted tree decomposition $\{T_x \mid x \in \mathcal{T}\}$ of width at most $3k + 3$ such that $W \subset T_r$ where r is the root of $\mathcal{T}$.

We prove this statement by induction on $|V|$. For $|V| \leq 3k + 4$, the graph has a tree decomposition with every vertex in one bag with the properties above.

Now suppose $|V| > 3k+4$. Let $W \subset V$ with $|W| = 2k+3$. Let S be a balanced W-separator of cardinality at most $k + 1$, and let $C_1, \ldots, C_m$ be the connected components of $G \setminus S$.

For $1 \leq i \leq m$, let $V_i = C_i \cup S$ and G_i the induced subgraph of G with vertex set V_i. Then we have

$$|V_i \cap W| \leq |C_i \cap W| + |S| \leq \frac{|W|}{2} + |S| \leq k + 1 + k + 1 < 2k + 3 = |W|\,.$$

Therefore, there is a vertex in W not in V_i, so $|V_i| < |V|$. Let $W_i = (C_i \cap W) \cup S$. We have $|W_i| \leq |C_i \cap W| + |S| \leq 2k + 2$. Since $|V_i| < |V|$, the induction hypothesis holds for each G_i, and for each $1 \leq i \leq m$, there exists a tree decomposition $\{T_x^i \mid x \in \mathcal{T}_i\}$ of width at most $3k + 3$ with W_i contained in the root. We join these tree decompositions together at a new root r with $T_r = W \cup S$ having an edge to each of the m other roots. It is easy to check that this is a tree decomposition for G. Since $|W \cup S| \leq 2k + 3 + k + 1 = 3k + 4$, the new tree decomposition has width at most $3k + 3$. □

The above theorem indicates that we can find treewdith by finding balanced W-separators. However, there are only efficient algorithms for finding separators of two sets, so instead of finding balanced separators, we look for weakly balanced separators, which we now define.

Definition 3.1.2. Let $G = (V, E)$ be a graph and $W \subset V$. A weakly balanced separation of W is a triple (X, S, Y) where $X, Y \subset W$, $S \subset V$ are pairwise disjoint sets such that $W = X \cup (S \cap W) \cup Y$, X and Y are disconnected from each other in $G \setminus S$, and $0 < |X|, |Y| \leq \frac{2}{3}|W|$.

Lemma 3.1.2 ([6])**.** *Let $k \geq 2$, $G = (V, E)$ be a graph of treewidth at most k, and $W \subset V$ with $|W| \geq 2k + 3$. Then there exists a weakly balanced separation of W of order at most $k + 1$.*

Proof. By Theorem 3.1.1(1), there exists some balanced W-separator S of cardinality at most $k+1$. Let $C_1, \ldots, C_m$ be the connected componnents of $G \setminus S$ with nonempty intersection $W_i = C_i \cap W$. Then $|W_i| \leq \frac{|W|}{2}$ for $1 \leq i \leq m$. Since $\bigcup_{i=1}^{m} W_i = W \setminus S$, and $|W \setminus S| \geq |W| - (k+1) > \frac{|W|}{2}$, we must have that $m \geq 2$.

Without loss of generality, assume that $|W_1| \geq |W_2| \geq \ldots \geq |W_m|$. Let $1 \leq i \leq m$ be the minimum number such that $\sum_{j=1}^{i} |W_j| > \frac{|W \setminus S|}{3}$. i is well-defined since $\sum_{j=1}^{m} |W_j| = |W \setminus S| > \frac{|W \setminus S|}{3}$. We let $X = \bigcup_{j=1}^{i} W_j$ and $Y = \bigcup_{j=i+1}^{m} W_j$. Clearly, $|X| > 0$ and $|Y| \leq \frac{2}{3}|W|$. Suppose $|W_1| > \frac{|W \setminus S|}{3}$. Then $i = 1$. Thus $|X| = |W_1| \leq \frac{|W|}{2} \leq \frac{2}{3}|W|$, and $|Y| > 0$ since $m \geq 2$. Now suppose $|W_1| \leq \frac{|W \setminus S|}{3}$. Then, by the ordering of the sets W_i, we also have that $|W_i| \leq \frac{|W \setminus S|}{3}$. Hence,

$$|X| = \sum_{j=1}^{i} |W_j| = \sum_{j=1}^{i-1} |W_j| + |W_i| \leq \frac{|W \setminus S|}{3} + \frac{|W \setminus S|}{3} = \frac{2}{3}|W \setminus S| \leq \frac{2}{3}|W|,$$

and $|Y| = |W \setminus S| - |X| > 0$. □

3.2 A 4-Approximation for Treewidth

Now we have the tools to outline a 4-approximation algorithm that finds the treewidth of a graph. We first use the fact that for a graph $G = (V, E)$, sets $X, Y \subset V$, and $k \in \mathbb{N}$, there is an algorithm running in time $O(k|G|)$ which decides whether there exists a set $S \subset V$ of at most k elements that separates X from Y and computes such a set if it exists. This is a well-known network flow algorithm that maintains a family of disjoint paths from X to Y and iteratively adds new paths to the family by finding alternating paths.

Using the above algorithm, we can construct a new algorithm which, given a graph $G = (V, E)$, pairwise disjoint sets $X, Y, Z \subset V$, and $k \in \mathbb{N}$, decides whether there exists a set $S \subset V \setminus (X \cup Y)$ such that $|S| \leq k$, $Z \subset S$, and S separates X from Y and computes such a set if there exists one. We define X' as all the vertices in $V \setminus X$ which are adjacent to X and define Y' similarly. We let G' be the induced subgraph of G with the vertices $V \setminus (X \cup Y)$. Then we only need to use the algorithm above on the graph G' to find a separator for $X' \cup Z$ and $Y' \cup Z$. Since Z is in both sets, it will have to be a subset of the output S since otherwise the sets would not be disconnected. Also the set S will be disjoint from X and Y since they are not in G'. Also it is clear that S is a separator for X' and Y' in G' if and only if S is a separator for X and Y in G and $S \cap (X \cup Y) = \emptyset$. Since the algorithm above takes time $O(k|G|)$, this one does too.

Lemma 3.2.1 ([6]). *There is an algorithm that solves the following problem in time $O(3^{3k}k\,|G|)$: Given a graph $G = (V, E)$, $k \in \mathbb{N}$, a set $W \subset V$ such that $|W| = 3k + 1$, decide whether there exists a weakly balanced separation of W of order at most $k + 1$ and compute such a separation if it exists.*

Proof. Because we need $0 < |X|, |Y| \leq \frac{2}{3}|W|$ for a weakly balanced separation, we must have that $0 < |X|, |Y| \leq 2k$. We can use an algorithm to check for all possible disjoint subsets $X, Y, Z \subset W$ such that $0 < |X|, |Y| \leq 2k$, $|Z| \leq k + 1$ and $W = X \cup Y \cup Z$ if there exists a weakly balanced separation for W with this X and Y by running the above algorithm with X, Y, Z, and $k + 1$. If the algorithm finds such and S, we have that $Z \subset S$ and S separates X from Y. The number of triples of sets we need to check is less than the number of ways to partition W into 3 sets. Since $|W| = 3k + 1$, there are 3^{3k+1} such partitions. For each partition we do some linear-time checks on the graph and then apply the above algorithm on the graph, which takes time $O(k\,|G|)$. Hence, the overall time of this algorithm is $O(3^{3k}k\,|G|)$. □

Theorem 3.2.2 ([6]). *There is an algorithm that, given a graph $G = (V, E)$ with treewidth $tw(G)$ finds that G has a tree decomposition of width k where $k \leq 4tw(G)+1$ in time $O(3^{3tw(G)}tw(G)\,|G|)$.*

Proof. Using the algorithm in Lemma 3.2.1, we can find a weakly balanced separation for a graph G. When we find a weakly balanced separation for a set W with $|W| = 3k + 1$, we can use the same construction as in Theorem 3.1.1(2) to create a tree decomposition of width $4k + 1$, where there the separator of cardinality $k + 1$ is used. Therefore, if we run the algorithm in Lemma 3.2.1 using k as input and find a separator, we know that G has treewidth at most $4k + 1$.

We can run an algorithm which chooses some $W \subset V$ such that $|W| = 3k + 1$ and sets $k = 1$ and then calls the algorithm in Lemma 3.2.1 with k as input. If a separator is found, $4k + 1$ is returned as a treewidth estimate. Otherwise, k is increased by 1 and the algorithm continues. The algorithm is guaranteed to terminate by Lemma 3.1.2 once k is set to $\mathrm{tw}(G)$.

The running time of each stage is $O(3^{3k}k\,|G|)$. Therefore, the total running time is

$$\sum_{k=1}^{\mathrm{tw}(G)} c3^{3k}k\,|G| \leq c \cdot \mathrm{tw}(G)\,|G| \sum_{k=1}^{\mathrm{tw}(G)} 3^{3k} = O(3^{3\mathrm{tw}(G)}\mathrm{tw}(G)\,|G|).$$

□

Chapter 4

Future for Treewidth

Even though finding the treewidth of a graph is NP-complete, the approximation algorithm outlined gives hope that in many likely cases, finding treewidth may still be practical. There are even faster and more accurate approximations that have been developed in recent years and are continuing to be developed right now.

Bibliography

[1] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

[2] Donald J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32, 1970.

[3] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowksi. Complexity of finding embeddings in a k-tree. *SIAM*, 8, Apr 1987.

[4] Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM*, 2, Mar 1981.

[5] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W. H. Freeman, 1979.

[6] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

Academic Vita

Daniel Hofman

38 Hamilton Hall, University Park, PA 16802
(302) 367-6079 • dsh5289@psu.edu

Education: **Undergraduate Senior at Pennsylvania State University**
Concurrent major in Computer Engineering and Mathematics

Courses:

Mathematics	**Computer Engineering**
Calculus I & II	Intermediate Programming
Vector Calculus	OOP with Web-based Applications
Differential Equations	Digital Design
Linear Algebra	Computer Organization
Abstract Algebra	Circuits and Devices
Analysis I & II	Systems Programming
Complex Analysis	Operating Systems
Numerical Analysis	Microelectronics
Probability Theory	Signals and Systems
Statistics Theory	Algorithms and Data Structures
Topology	Theory of Computation
Graph Theory	Algorithm Design and Analysis
Functional Analysis	Complexity of Combinatorial Problems

Honors: Dean's List at Pennsylvania State University each semester
Evan Pugh Senior Scholar Award
Chris Mader Memorial Scholarship
Mary Lister McCammon Award for Mathematics

Research: **Musical Minds** (Summer 2016 – Fall 2016)
I worked as a lead software developer for Musical Minds, a startup being run out of Penn State. I researched using machine learning to predict mental states from EEG signals and collaborative filtering to recommend new music to users based on their past ratings of songs.

Laboratory for Perception, Action, and Cognition (Fall 2016 – Spring 2017)
I contributed to research under Dr. Yanxi Liu in Penn State's Laboratory for Perception, Action, and Cognition (LPAC), which focuses on research in computer vision. I programmed a reCAPTCHA website to collect data from humans on reflection and rotation symmetries in images.

Combinatorics and Algorithms for Real Problems REU (Summer 2018)
Combinatorics and Algorithms for Real Problems (CAAR) was a Research Experience for Undergraduates (REU) at the University of Maryland. Under Drs. Clyde Kruskal and Tom Goldstein, I researched the chromatic number of the plane problem by trying to find proper colorings of the plane using computational approaches involving simulated annealing and SAT solvers.