NON-COMMON OPTIMALITY MEASURES FOR TREE-DECOMPOSITIONS

FERNANDO SÁNCHEZ VILLAAMIL

Diplomarbeit

August 2011

DECLARATION

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, August 2011

Fernando Sánchez
Villaamil

ABSTRACT

Normally the quality of a tree-decomposition is measured by its treewidth. This suffices to bound the running time of algorithms that works on tree-decompositions. Nevertheless the presence of join bags, both their number and their size, can have a very big impact on the running time of such an algorithm on specific instances. In this thesis we deal with the analysis of the structure of tree decompositions, not only their width. We will first define *tight asteroid triples*. We will then link the appearance of this structure in the chordalization given by a tree decomposition with the number of joins that this tree decomposition must have. Furthermore we will present a heuristic to minimize the size of join bags and some ideas about how to tackle related problems based on the notions established in this thesis.

ZUSAMMENFASSUNG

Normalerweisse wird die Güte einer Baumzerlegung an seiner Baumweite gemessen. Dies reicht in den meisten Fällen aus, um eine obere Grenze für die Laufzeit eines Algorithmus, welcher auf Baumzerlegungen arbeitet, zu finden. Die Anwesenheit von *join*-Taschen in der Baumzerlegung jedoch kann einen sehr großen Einfluss auf die Laufzeit eines solchen Algorithmus für eine spezifischen Instanz haben. Dabei ist sowohl die Größe als auch die Anzahl der *join*-Taschen ausschlagegebend. In dieser Arbeit werden wir uns mit der Analyse der Struktur von Baumzerlegungen beschäftigen. Dazu führen wir das Kozept von *tight asteroid triple* ein. Wir werden dann die Häufigkeit dieser Struktur in der Chordalisierung eines Graphens, die von einer Baumzerlegung gegeben ist, mit der Anzahl der *joins*, die diese Baumzerlegung haben muss verknüpfen. Darüber hinaus beschreiben wir eine Heuristik, um die Größe der *join* Taschen zu minimieren. Abschließend erörtern wir weitere Ideen wie man die Konzepte dieser Arbeit benutzen könnte, um verwandte Probleme zu lösen.

ACKNOWLEDGMENTS

Many thanks to Somnath, Felix, Alexander, Joachim and Prof. Rossmanith for always having an open ear. Thanks to Birgit for forcing me to have at least a modicum of organization in my life. Finally thanks to all my friends for dealing with my stressed self.

CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

1 INTRODUCTION

When the concept of NP-hardness was developed in the 70's [20], it was generally assumed that exact solutions for instances of NP-hard could not be calculated in a reasonable time. NP-hardness was mainly used as a framework to decide if it was sensible to develop an exact algorithm for the problem or not. When a solution for an NP-hard problem was needed in practice, either an approximation algorithm or a heuristic was used to find a solution that was in many cases "good enough". This approach is not always satisfactory. In many applications nothing but an exact solution is acceptable, e.g. route-planning, where the costs inflicted by a non-optimal solution can be very large. A particularly relevant field in this respect is computational biology, where for some problems approximations are bad models of reality, and as such are not useful to run inexpensive simulations of experiments instead of the costly experiments themselves. Such problems are one of the main reasons for the growing interest in exact algorithms for NP-hard problems. This together with the development of Parameterized Complexity Theory by Downey and Fellows [17], which deals with the design and analysis of exact algorithms for NP-hard problems, has given new life to the field of exact algorithms.

The notion on which Parameterized Complexity Theory is based is that not all instances of an NP-hard problem are difficult to solve. There are simple exact algorithms that work very well for many, if not most instances of a problem. Often the instances that arise in practice are a pretty restricted subclass of all the possible instances. The problem may still be NP-hard on this subclass, the restrictions can nevertheless help us design a special algorithm adapted to this subclass that is fast enough for our purposes. The natural question that arises is then: What are the properties of an instance that make it hard to solve? The general idea is to take this property, relate it to a parameter k which is, for lack of a better word, a representation of the property and then try to find a so-called *fixed-parameter tractable* algorithm for it. An algorithm is fixed-parameter tractable if its running time can be bounded by $O(f(k) \cdot poly(n))$, where n is the size of the input, $poly(n)$ is a polynomial, k is the so-called *parameter* (which also must be part of the input) and f is an arbitrary function (f does not even have to be decidable). This is the most general case. For all practical situations f is a computable and slowly growing exponential function. A prob-

lem is said to be fixed parameter tractable (or in FPT) if there exists a fixed-parameter tractable algorithm for it. If we have a fixed-parameter tractable algorithm for a problem on a specific parameter k it follows that all elements in a set of instances of the problem all belonging to a class where k is fixed can be solved in polynomial time. Even if k is not fixed, if this parameter is small, and the function $f(k)$, the polynomial over n and the hidden constants in the big O notation are not too bad, then it is still guaranteed that the algorithm will find a solution in a reasonable time.

The standard parameter is the size of the solution. This means that if an instance has a small solution, then it can be computed in a short time. Even though this is the most common parameter, and in many cases it is also the most natural. Well known problems like Independent Set and Dominating Set probably do not admit a fixed-parameter tractable algorithm under reasonable assumptions if we take the size of the solution as the parameter [15, 16]. Since we are free to chose a different one it seems then logical to look for different useful parameters. One such parameter is *treewidth*.

For a fixed-parameter tractable algorithm for which treewidth is the parameter we assume that a *tree-decomposition* of treewidth k is given as part of the input. We can assume this because calculating a tree-decomposition of width k is fixed parameter tractable, with k as a parameter [8]. The treewidth of a graph is given by the existence of a tree-decomposition of a graph with that width. A tree-decomposition of a graph G is composed by a set of bags $\{X_i \mid i \in I\}$ with a labeling I and a tree $T = (I, E)$. This structure has to have the following characteristics.

- All nodes of the graph must be contained in at least one bag.
- If $(u, v) \in E(G)$ then there must be at least one bag B such that $u, v \in B$.
- If there are two bags B and B' in the tree-decomposition that both contain u, all bags in the in T path between B and B' must contain u too.

The width of a tree-decomposition is the size of the biggest bag minus one. The width of a graph G is the minimum width of all possible tree-decomposition of G. It is often said that the width describes how tree-like a graph is, e.g. the width of a tree is one and the width of a complete graph is the number of nodes minus one. In recent times there has been a growing interest in the study of algorithms on tree-decompositions. One of the main reasons for this interest is Courcelle's proof that all problems

definable in monadic second order logic are fixed-parameter tractable when treewidth is chosen as the parameter [14].

Bags of degree one are leaves, bags of degree two are *introduce* bags and bags of degree greater than two are *join* bags. Algorithms which work on tree-decomposition normally work on a special kind of tree-decompositions called *nice tree-decompositions*. On top of the properties previously mentioned the following ones must also be met in nice tree-decompositions:

- The tree must be rooted.
- The tree is a binary tree.
- All leaves must contain a single node.
- If (B, B') is an edge in the tree-decomposition then $|B - B'| \leq 1$.
- All the children (not the parent) of a join bag must be the same as the join bag.

Every tree-decomposition can be converted into a nice tree-decomposition in polynomial time [10]. This makes it easier to design a dynamic programming algorithm on tree-decompositions. Such an algorithm normally works by creating tables for the leaves of the decomposition and then updating these tables moving from the leaves of a tree-decomposition to its root:

- First create a table for the the leaves.
- Then "move" all tables towards the root:
  - If the parent of the current bag is an introduce bag, then create the table for this parent bag by updating the table using a specific operation for introduce bags.
  - If the parent is a join, then we need the table for the other child of the join bag too. Create the table for the join bag by "merging" the tables of its two children.
- The solution can be found in the table of the root bag.

The details of how this is implemented are not very important. What is important is to see that these algorithms can have different operations for introduce bag and join bags. Most actually have. Normally, the running time of the algorithm can be bounded by the size of the bags (the width of the tree-decomposition), but it is clear that the running time of such an algorithm is affected by the structure of the tree and not only by the size of the bags. There are algorithms where e.g. the operation on introduce bags requires just to update the table

row by row, possibly creating a constant number of extra rows for each already existing row, while the operation on join bags requires every row in a table of a child of the join to be compared every other row in the table of the other child of the join. Even if the upper bound of the running time of the operations on join and introduce bags is the same, the hidden constants and the memory requirements are generally worse for join bags.

In this thesis the problem of the structure of tree-decompositions is dealt with. We will assume that manipulating dynamic programming tables for join bags is costlier than for introduce bags. We can then abstract from the actual implementation of the algorithm and concentrate on optimizing a tree-decomposition on the two accounts that have an impact on the running time:

- The number join bags in a nice tree-decomposition of the graph.
- The size of these join bags.

We will mainly deal with the following question: What is the minimum number of joins necessary in a natural tree-decomposition of a chordal graph? A chordal graph is a graph that contains no chordless cycles of length greater than three. A natural tree-decomposition is one where the content of every bag is a clique in the graph. The class of graphs that have natural tree-decompositions is precisely the class of chordal graphs.

There is a good reason why this is a reasonable way to begin to tackle the problem. It is known that every tree-decomposition gives a chordalization of the graph. A chordalization of a graph G is a set of edges that when added to G result in a chordal graph. Even more, the minimum tree-decomposition (optimality measured on width) gives the minimum chordalization of the graph (number of edges added). That means that either explicitly or implicitly calculating a tree-decomposition of general graphs means calculating a chordalization of the graph. So this is a special case of the general problem of finding optimal tree-decompositions weighting the number of joins and the size of introduce nodes. By analyzing what forces there to be joins in the natural tree-decompositions of the underlying chordalization of a tree-decomposition, we can investigate what causes there to be joins in tree-decompositions of general graphs.

We will find that the necessary number of joins we need in any case can be bounded by a specific structure in the chordalization given by this tree-decomposition. It is not improbable that this results give a good footing for further research on the structure of tree-decompositions. Furthermore we will present an heuristic to minimize the size of the joins bags, given a tree-decomposition, and we will present an idea to improve the size

of the join bags in a given tree-decomposition at the cost of increasing the size of some introduce bags.

To the best of our knowledge, no structural result that bounds the number of joins necessary in a tree-decomposition was known. Almost all the literature on tree-decompositions uses the width as the sole measure of optimality. Hence there is a dearth of tools to deal with the structure of tree decompositions, even though the structure can have a big impact on the running time of a dynamic programming algorithm. Further work on the results of this thesis may change that. To the best of our knowledge heuristics, to minimize the number and the size of joins have never been formalized. These heuristics may be very useful in practice.

Here is a brief description of the organization of this thesis: In Chapter 2 we will give some necessary definitions and present known results related to this area. In Chapter 3 we will present our results. Finally in Chapter 4, we will further elaborate on the results of this thesis and their implications, and give ideas for future research.

2

DEFINITIONS AND IMPORTANT KNOWN RESULTS

In this chapter we give all the necessary definitions and facts needed to understand the results of Chapter 3.

2.1 DEFINITIONS AND BASIC RESULTS

Given sets A and B we let $A + B$ denote the union $A \cup B$ and $A - B$ denote the complement $A \setminus B$. If $\mathcal{M}$ is a collection of sets, then $\bigcup \mathcal{M} = \{e \in M \mid M \in \mathcal{M}\}$.

A graph is denoted by $G = (V, E)$, where V is the set of nodes in the graph and E is the set of edges. All graphs we will deal with are finite, simple and undirected and, unless noted otherwise, are connected. $V(G)$ will denote all the nodes in the graph G and $E(G)$ all its edges. For $V' \subseteq V(G)$ we let $G[V']$ denote the graph with vertex set V' and edge set $(V' \times V') \cap E(G)$. The *neighborhood* of a node $v \in V(G)$ will be denoted as $N(v)$ and the *closed neighborhood* as $N[v] = N(v) + \{v\}$. The degree of a node v in a graph is defined as $d(v) = |N(v)|$.

A path $p = (p_1, \ldots, p_l)$ in a graph G, is a sequence of vertices of the graph such that $p_i \neq p_{i+1}$ and $(p_i, p_{i+1}) \in E(G)$ for all $1 \leq i \leq l - 1$. A cycle is a path where $p_1 = p_l$. We will abuse the notation such that one of the elements in a path may be another path. If $(p_1, \ldots, p_l)$ is a path such that for $1 \leq i \leq l$ the element $p_i = (p'_1, \ldots, p'_j)$ is a path, then the path we are referring to is $(p_1, \ldots, p_{i-1}, p'_1, \ldots, p'_j, p_{i+1}, \ldots, p_l)$. If $p = (p_1, \ldots, p_l)$ is a path we will denote the reversed path as $p^R = (p_l, \ldots, p_1)$.

We will use the same notation we use for paths for pairs and triples. If u, v and w are nodes in a graph, then (u, v) or (u, v, w) must not be paths, they may just be a pair and a triple of nodes respectively. What is meant will always be clear from the context.

A *partition* of of a set S is a set $\mathcal{P} = \{P_1, \ldots, P_k\}$, such that $\bigcup \mathcal{P} = S$ and for every pair $P, P' \in \mathcal{P}$, $P \cap P' = \emptyset$. An element of $\mathcal{P}$ is called a *cell*.

A $\text{star}_{k,k,k}$ will denote a graph that is composed by three paths $(u, u_1, \ldots, u_k)$, $(u, u'_1, \ldots, u'_k)$ and $(u, u''_1, \ldots, u''_k)$ such that all nodes in $\{u_1, \ldots, u_k, u_1, \ldots, u'_k, u''_1, \ldots, u''_k\}$ are pairwise distinct and there are no further edges in the graph (see Figure 3.1).

This thesis is about tree-decompositions, so let us define it formally.

Definition 2.1 (Tree-Decomposition)**.** *A tree-decomposition $\mathcal{T}$ of a graph G is a pair $(\{X_i \mid i \in I\}, T = (I, E))$ where the sets X_i are subsets of $V(G)$, are called bags and are denoted by $\mathcal{B}(\mathcal{T})$ and T is a tree such that:*

- $\bigcup_{i \in I} X_i = V(G)$.
- *If $(u, v) \in E(G)$ then there exists $i \in I$ bag such that $u, v \in X_i$.*
- *For all $i, j, k \in I$ where j is in the path from i to k in T, $X_i \cap X_k \subseteq X_j$.*

Note that every graph G has a trivial tree-decomposition $(\{X_1 = V(G)\}, T = (\{1\}, \emptyset))$. The *treewidth* of a tree-decomposition is the size of the biggest bag minus one. The treewidth of a graph is the minimum treewidth over all valid tree-decomposition of the graph. Since the bags of a decomposition are the nodes of a tree we may treat them both as sets as well as vertices of a graph. Hence all operations on sets and graph nodes are valid on the bags of a tree-decomposition.

We will often use a very well known result about tree-decompositions.

Proposition 2.2 ([3])**.** *Let $\mathcal{T} = (\{X_i \mid i \in I\}, T = (I, E))$ be a tree-decomposition of a connected graph and let $B \in \mathcal{B}(\mathcal{T})$ be one of its bags. If B is an inner node and we remove it, then the tree falls apart into two components. If u is a node that is just in one of these components and v is one that is just contained in the other, then B is a separator of u and v.*

We will use this to prove some things more easily than through the details of the tree-decomposition definition.

When designing algorithms on tree-decompositions it is often assumed that the tree-decomposition is a *nice tree-decomposition*.

Definition 2.3 (Nice Tree-Decomposition)**.** *A* nice tree-decomposition *$(\{X_i \mid i \in I\}, T = (I, E))$ of a graph G is a tree-decomposition where T is a rooted binary tree such that the following properties hold:*

- *If $i \in I$ is a leaf in T, then $|X_i| = 1$.*
- *Every node $i \in I$ maps either to an introduce, a forget or a join bag:*
  - *X_i is an introduce bag if i has only one child j where $X_i = X_j + \{u\}$ and $u \in V(G)$.*
  - *X_i is a forget bag if i has only one child j where $X_i = X_j - \{u\}$ and $u \in V(G)$.*
  - *X_i is a join bag if i has exactly two children j and k and $X_i = X_j = X_k$.*

Proposition 2.4 ([10]). *Every tree-decomposition can be converted into a nice tree-decomposition with the same treewidth in polynomial time.*

We need to describe further special subsets of tree-decompositions.

Definition 2.5 (Natural Tree-Decomposition). *A* natural tree-decomposition *of a graph G is one in which if two nodes u, v of G are in the same bag, then $(u, v) \in E(G)$.*

We will see that the only graphs that have a natural tree-decomposition are chordal graphs. There are many characterizations of chordal graphs, but we will use only three of them. First we will need to define what a *perfect elimination order* is.

Definition 2.6 (Perfect Elimination Order). *A graph G has a* perfect elimination order *iff there is an ordering $\{u_1, \ldots, u_n\} = V(G)$ of all the nodes in G such that for every $1 \leq i < n$ the set $\{u_{i+1}, \ldots, u_n\} \cap N(u_i)$ is a clique.*

Definition 2.7 (Chordal Graph). *A graph G is chordal iff any one of the following conditions hold:*

- *G has no chordless cycle of length greater than three.*
- *G has a natural tree-decomposition (see [18]).*
- *G has a perfect elimination order (see [3, 25]).*

Proposition 2.8 ([25]). *We can find a perfect elimination order of a chordal graph G in time $O(n^3)$. This gives us all the maximal cliques in G.*

Definition 2.9 (Chordalization). *A* chordalization *of a graph $G = (V, E)$ is a set of edges $E' \cap E = \emptyset$ such that $G' = (V, E + E')$ is chordal.*

The size of a chordalization is measured by number of edges that are added. A minimum chordalization is then one that adds a minimum number of edges. A tree-decomposition of minimum width of a graph G also gives us a chordalization of G.

Proposition 2.10 ([24]). *If $G = (V, E)$ is a chordal graph and $\mathcal{T}$ is a tree-decomposition of G then $G' = (V, E + \{(u, v) \mid u, v \in B,\ B \in \mathcal{B}(\mathcal{T})\})$ is a chordalization of G.*

In this thesis, we will redefine the meaning of join, introduce and forget nodes (see Definition 2.3) to better suit our needs. Before we had only defined these concepts for nice tree-decompositions, but for our purposes it is not easy to work with nice tree-decompositions.

Definition 2.11 (Join Bag)**.** *In a tree-decomposition a* join bag *is a bag whose degree is greater than two. For a tree $\mathcal{T}$ the set of all join bags will be $\mathcal{J}(\mathcal{T})$.*

Definition 2.12 (Introduce Bag)**.** *Any bag of a tree-decomposition that is not a join bag is an* introduce *bag. The* set of introduce bags *of a tree-decomposition $\mathcal{T}$ will be $\mathcal{I}(\mathcal{T}) = \mathcal{B}(\mathcal{T}) - \mathcal{J}(\mathcal{T})$.*

Not that we do not distinguish between forget and introduce nodes. There is no definition of a forget bag. Instead of working with nice tree-decompositions, we are going to look at tree-decompositions that handle the maximal cliques of chordal graphs.

Definition 2.13 (Maximal Clique Tree-Decomposition)**.** *A* maximal clique tree-decomposition *is a natural tree-decomposition of a chordal graph, where all the bags are maximal cliques, except the join bags, and no bag appears more than once.*

Definition 2.14 (Strict Maximal Clique Tree-Decomposition)**.** *A* strict maximal clique tree-decomposition *is a natural tree-decomposition of a chordal graph, where all the bags are maximal cliques, and no bag appears more than once.*

Lemma 2.15. *All chordal graphs have a natural strict maximal clique decomposition.*

Proof. It is a well known result that all maximal cliques of a graph must be contained in any tree-decomposition of the graph. Since in a natural tree-decomposition all bags are cliques, all bags must either contain a maximal clique or a subset thereof. From [6] we know that we can convert any tree-decomposition into a tree-decomposition such that for any two bags B and B', $B' \nsubseteq B$. From this the lemma follows. □

Corollary 2.16. *All chordal graphs have a natural maximal clique decomposition.*

In this thesis we analyze the structure of tree-decompositions. This means that we may have to make some assumptions about how the running time of these algorithms relates to the structure of the tree-decompositions. Our results would improve the running time of algorithms that fall under the definition of a *normal tree-decompositions algorithm*.

Definition 2.17 (Normal Tree-Decomposition Algorithm)**.** *A algorithm that works on tree-decompositions is* normal *if it works on any nice tree-decomposition $\mathcal{T}$ of a graph G with n nodes, and there exists an introduce function $i\colon \mathbb{N} \times \mathbb{N} \mapsto \mathbb{R}$ and an join function $j\colon \mathbb{N} \times \mathbb{N} \mapsto \mathbb{R}$, where for all $x, n' \in \mathbb{N}, j(x, n') \geq i(x, n')$, such*

that for an instance $\mathcal{T}$ the running time of the algorithm is bounded by

$$\sum_{X \in \mathcal{I}(\mathcal{T})} i(|X|, n) + \sum_{X \in \mathcal{J}(\mathcal{T})} j(|X|, n)$$

The running time of a normal tree-decomposition algorithm on a specific nice tree-decomposition will depend on the number of join and introduce bags. We said before that we are not going to work with nice tree-decompositions, that is why we need to define a measure for general tree-decompositions.

Definition 2.18 (Number of Joins)**.** *The number of joins of a tree-decomposition $\mathcal{T}$ is:*

$$\sum_{X \in \mathcal{J}(\mathcal{T})} (d(X) - 2)$$

Notice that we distinguish between the number of join bags and the number of joins in a tree-decomposition. The number of joins in any tree-decomposition $\mathcal{T}$ is precisely the number of join bags in a nice tree-decomposition $\mathcal{T}'$ obtained from $\mathcal{T}$.

2.2 LITERATURE SURVEY

Bodlaender and Fomin investigate in [6] the structure of tree-decompositiona, but only in the sense that the size of all bags are taken into account, not just the size of the biggest one. They relate this to a function $f : \mathbb{N} \mapsto \mathbb{R}$ whose value correlates with the running time of an algorithm on a bag, given the size of the bag. The measure they optimize for a tree-decomposition $\mathcal{T}$ is then:

$$\sum_{B \in \mathcal{B}(\mathcal{T})} f(|B|)$$

As long as the function $f(n)$ is in $\Omega(2^n)$ they proof that the optimal tree-decomposition is one of the minimum chordalization of the graph. We saw before that all chordalizations of a graph give a tree-decomposition of the graph. This means that we can find the optimal tree-decomposition on the measure previously given by enumerating all minimum chordalizations. Since all minimum chordalizations of a graph can be determined by finding maximal cliques in its separator graph [23], they construct a polynomial time algorithm for graph classes with a polynomial number of minimum separators. This way of measuring optimality is closer to the actual running time an algorithm will have on a specific tree-decomposition, but it still does not distinguish between join bags and introduce bags.

There were two interesting results on which we can build on to tackle our problem. Both are based on the notion of *asteroidal triples*. We will define what an asteroidal triple is in Chapter 3.

One result that can be expanded upon to solve our problem is given in [27]. Here a characterization of tree-decompositions that have a small pathwidth is given. In this paper the measure of *catwidth* is defined, which is the width of a tree-decomposition of pathwidth one. This is an interesting measure because one can show that the memory requirements for such tree-decompositions are lower for many algorithms. A class of graphs where the tree-decompositions of the graphs have this property is given. This is done by extending the definition of asteroidal triples. Such a result goes in the right direction for our purposes, since it is a result about the structure of tree-decompositions, even if it is a very narrow one. We will make a more generalized extension of asteroidal triples to talk about the number of joins in tree-decompositions of any pathwidth.

There are other indications that asteroidal triples are a good concept on which to base an analysis of this problem. It is a known result that there is a tree-decomposition of minimum width for asteroidal triple free graphs that is a path decomposition [21]. Also, graph classes with a bounded number of asteroidal triples have a constant factor approximation algorithm for treewidth (the factor depends on the number of asteroidal triples)[11]. The problem is that the classes of graphs that are known to be asteroidal triple free graphs seems to be somewhat restricted, e.g. having to contain a pair of nodes such that all paths joining them are dominating sets of the graph [12, 13].

To the best of our knowledge nobody has previously attempted to solve the problem of minimality of number and size of joins in chordal graphs.

3

BASIC TREE STRUCTURE

Our first objective when dealing with the structure of tree-decompositions will be to find tree-decompositions of minimum width with a minimum number of join nodes. This would allow us to minimize the running time even more than we would by just minimizing treewidth. This problem is of course NP-complete, since it must be even harder than finding a tree-decomposition of minimum width.

We know that any tree-decomposition of minimum width implicitly specifies a minimum chordalization of a graph. We also know that calculating a tree-decomposition for a chordal graphs can be done in polynomial time. If we would solve the general problem of minimumity of both width and number of joins for general graphs, we would thus implicitly solve the easier problem of minimum number of joins for chordal graphs. In this chapter we are mainly going to look at this reduced problem:

MINIMUM NUMBER OF JOINS ON CHORDAL GRAPHS

Input: A chordal graph G.

Problem: Find the minimum number of joins of any natural tree-decompositions of G.

All natural tree-decompositions of G are of minimum width. We could restate this problem as: Given a chordal graph G, find a tree-decomposition of G with a minimum number of joins that maintains the known minimum width. Finding a solution for this problem would give us a two step heuristic for the general problem:

1. Find some tree-decomposition of the graph G with some of the already known algorithms that minimizes width.
2. Attempt to optimize the given tree-decomposition to decrease the number of joins.

More specifically this would have the following work-flow:

1. Find some tree-decomposition of the graph G with some of the already known algorithms.
2. Calculate the chordalization G' of G given by the tree-decomposition.

3. Try to find a natural tree-decomposition of G' with a minimum number of joins.

The interest in this subproblem goes beyond this heuristic. The hope is that by investigating what forces the existence of joins in the underlying chordalization given by any tree-decomposition algorithm we might be able to control for this while constructing tree-decompositions for general graphs.

In this chapter we are going to first deal with the problem of finding a natural tree-decomposition with a minimum number of joins for chordal graphs. At the end of the chapter we are also going to give a heuristic to minimize the size of these joins.

In Chapter 2 we stated as a previously widely known result that in a natural tree-decomposition of a chordal graph all bags are cliques in the graph. We also stated that all maximal cliques of a graph must be contained in some bag. This means we can restate our problem like this:

MINIMUM NUMBER OF JOINS ON CHORDAL GRAPHS

Input: All the maximal cliques $\mathcal{C}$ of a chordal graph G.

Problem: Find a natural tree-decomposition of G that connects bags containing each a clique in $\mathcal{C}$.

We will see that stating the problem in this fashion simplifies reasoning about it. This way of looking at the problem does not impose a big increment on the running time of any algorithm we would develop to find a good natural tree-decompositions, since we can calculate all the maximal cliques of a chordal graph in $O(n^3)$ time by finding the perfect elimination order of the graph and then removing all cliques that are a subset of other cliques (see Proposition 2.8).

The other thing we care about besides number of joins is their size. Ideally we would like to find a natural tree-decomposition where the added size of all the joins is the smallest. This would have the lowest impact on the running time. Expressed formally, the problem we ideally would want to solve is:

MINIMUM ADDED JOIN SIZE

Input: A chordal graph G.

Problem: Find a natural tree-decomposition $\mathcal{T}$ of G such that $\sum_{X \in \mathcal{J}(\mathcal{T})} (d(X) - 2) \cdot |X|$ is minimum.

In this chapter we are first going to try to minimize the number of joins. Doing this we are going to find a certain structure

in the underlying chordalization of any tree-decomposition that we can link to the number of joins. At the end of the chapter we are going to build on some minor results of this first part to find a heuristic to minimize the size of the joins, given a tree-decomposition, while mantaining its width.

To analyze these problem we will first prove some related structural results about tree-decompositions, specifically the nature of joins nodes in a tree-decomposition of a chordal graph. There are chordal graphs that have tree-decompositions that do not have join bags. A tree-decomposition of a graph that has no join bags, is a called a *path decomposition*. This raises the question of what the difference is between chordal graphs which have a natural path decomposition and those which do not. On the basis of such a characterization, we will try to link some structure to be found in chordal graphs with the number of joins any natural tree-decomposition of a chordal graph has to contain. The notion of *asteroidal triples* suffices to characterize these.

Definition 3.1 (Asteroidal Triple)**.** *An* asteroidal triple *is a triple* (x, y, z) *of pairwise distinct non-adjacent nodes such that for every pair of distinct nodes* $u, v \in \{x, y, z\}$ *there is a* u*-*v *path that does not contain any neighbor of the third node in the triple.*

The following result states formally the relation between asteroidal triples and natural path decompositions.

Proposition 3.2 ([22])**.** *A graph is an interval graph (has a natural path decomposition) iff the graph is chordal and asteroidal triple free.*

Therefore, if there is an asteroidal triple in a chordal graph G then any natural tree-decomposition of G has at least one join bag. Proposition 3.2 does not indicate *where* or how many joins are necessary. The natural question that arises is whether there is a connection between the number of asteroidal triples and the minimum necessary number of joins. This is not the case and it can be shown with an example: A $\text{star}_{k,k,k}$ graph will have at least $k - 1$ asteroidal triples (with no node in common), but it has a tree-decomposition that contains just one join bag (See Figure 3.1). Clearly an arbitrary number of asteroidal triples can "share" a separator in a tree-decomposition. But which are the ones that can not "share" a separator and which are the ones that can? To that end let us try to analyze what share means in this context. We will try to link asteroidal triples in graphs to asteroidal triples in tree-decompositions.

The first thing to notice is that since the nodes of asteroidal triples form an independent set, two of them can not be contained in a bag of a natural tree-decomposition. Thus the nodes of asteroidal triples must be distributed over several bags.

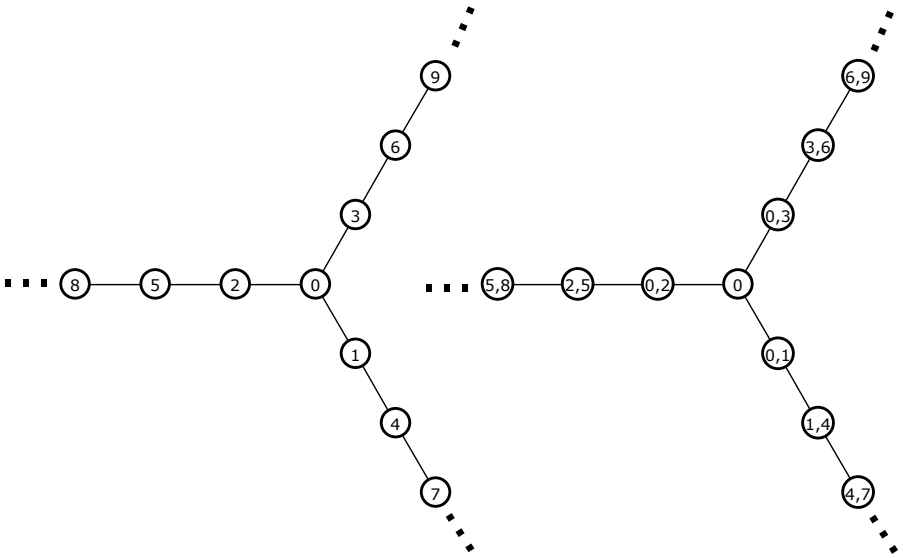

Figure 3.1: A $\text{star}_{k,k,k}$ and its tree-decomposition.

Definition 3.3. *We will say that three bags X, Y, Z* contain the asteroidal triple *(x, y, z) if $x \in X, y \in Y$ and $z \in Z$.*

With the help of this definition, the location of join bags in any tree-decomposition can be related to the bags that contain asteroidal triples.

Lemma 3.4. *Let $\mathcal{T}$ be a tree-decomposition of a chordal graph G and let X, Y and Z be bags in $\mathcal{T}$, whose content are maximal cliques of G. If X, Y, Z contain an asteroidal triple there is a join bag J that lies in every path in $\mathcal{T}$ between two of these bags.*

Proof. Assume the contrary. W.l.o.g. assume that the bag Y is the one that lies in a path from X to Z. For the asteroidal triple to be valid, there must be a path between x and z that does not intersect any node in Y, since $Y \subseteq N[y]$, as y is in the clique. Let p be any such path. In this configuration Y is a separator of X and Z, which means that it is also a separator of x and z. This implies that no such path p can exist, which would mean that (x, y, z) is not an asteroidal triple. □

We have deduced this far that if we have three maximal cliques in our chordal graph, such that they contain an asteroidal triple, the bags containing these maximal cliques must be connected through a join at some point.

It is clear that there are maximal cliques that contain different asteroidal triples but nevertheless can share a join bag in a natural tree-decomposition. Again, in our $\text{star}_{k,k,k}$ example, which can be seen in Figure 3.1, we can see that the tree-decomposition has just one join for all the asteroidal triples in the graph. The investigations in the next section will be based on this intuition.

3.1 NUMBER OF JOINS IN A NATURAL TREE-DECOMPOSITION

First we are going to investigate what the minimum necessary number of joins is. To make things easier, we are going to show that it is enough to look at tree-decompositions where all bags are maximal cliques, to solve this problem.

Lemma 3.5. *For a chordal graph G the minimum number of join bags of strict maximal clique decomposition[1] is the same as for natural tree-decompositions.*

Proof. Let $\mathcal{T}$ be a natural tree-decomposition of G and let J be a join bag in $\mathcal{T}$. Since it is a natural tree-decomposition, we know that $G[J]$ is a clique. As such $G[J]$ is either a maximal clique or a subset of at least one maximal clique J' of G. The maximal clique J' must be contained in some bag of $\mathcal{T}$. J must be connected to J' and all bags in the path between them must be a superset of J. We can create a strict maximal clique decomposition that has the same number of join bags by performing the following operations repeatedly until there are no more changes:

1. Find a bag J which is not a maximal clique. If no such bag exists we are done.
2. Let J' be the nearest bag in $\mathcal{T}$ containing a maximal clique which is a superset of J. Substitute the content of J with the content of J'.
3. Contract all edges in the new tree-decomposition that connect two bags with the same content.

The contraction operation does not increase the number of joins, which means that this conversion will give us a strict maximal clique decomposition with the same number of joins as before.

The other direction is trivial, since strict maximal cliques decompositions of chordal graphs are natural tree-decompositions. □

Remark 3.1. *There is a mapping between strict and non-strict maximal clique decompositions.*

We will use this notion of converting between strict and non-strict maximal clique decompositions later for a heuristic that tries to minimize the size of the join bags.

Having obtained a simplified version of the problem, let us investigate which maximal cliques we will have to use as joins. We will prove that there is a direct relation between a special kind of asteroidal triples (which we will call *tight asteroidal*

1 All bags must contain maximal cliques.

triple) and the number of join bags necessary in any natural tree-decomposition of a chordal graph. We first need another definition.

Definition 3.6 (Validating Path)**.** *A* validating path *is a shortest path between any two nodes of an asteroidal triple that does not cross the closed neighborhood of the third node.*

Notice that there can be more than one validating path between two nodes of a triple.

Definition 3.7 (Tight Asteroidal Triple)**.** *An asteroidal triple* (x, y, z) *is tight if no node* u *in a validating path of the triple forms an asteroidal triple* (u, v, w) *where* $u \notin \{x, y, z\}$ *and* $v, w \in \{x, y, z\}$*.*

We will be able to show that non-tight asteroidal triples are not necessary to bound the minimum number of joins necessary in a natural tree-decomposition. Any tree-decomposition of a chordal graph has to contain enough join bags to separate all asteroidal triples, so let us find a connection between tight and non-tight asteroidal triples over their separators.

Definition 3.8 (Separator of an Asteroidal Triple)**.** *The* separator of an asteroidal triple (x, y, z) *of a graph* G *is a set of nodes* $S \subsetneq V(G)$*, that separates each distinct pair* $u, v \in \{x, y, z\}$ *in the triple.*

First we need to proof something about cycles in chordal graphs. We will now specify where chords must appear in cycles of chordal graphs.

Lemma 3.9. *In any cycle* $C = (\dots, u, x, v, \dots)$ *of a chordal graph* G*, if* x *is not connected to another node in the cycle besides* u *and* v *then* $(u, v) \in E(G)$*.*

Proof. Let C' be an induced cycle of the graph that contains u, x and v and let $V(C') \subseteq V(C)$. If there is no edge between u and v this cycle has clearly length greater than three, which would mean that G is not chordal. □

Lemma 3.10. *An asteroidal triple* (x, y, z) *in a chordal graph is either tight or there exists a tight asteroidal triple* (x', y', z') *such that each node* $u \in \{x', y', z'\}$ *is either a node of* (x, y, z) *or lies in a validating path of* (x, y, z)*. We will call* (x', y', z') *a* corresponding tight asteroidal triple *of* (x, y, z)*.*

Proof. We will show that we can find the corresponding tight asteroidal triple of an asteroidal triple (x, y, z) in the following fashion:

1. If (x, y, z) is a tight asteroidal triple we are done.

2. If (x, y, z) is not tight, find a node u in a validating path that forms an asteroidal triple (u, v, w) with $v, w \in \{x, y, z\}$. Repeat step 1 for (u, v, w).

This would always yield a corresponding tight asteroidal triple if we can show that in chordal graphs this loop ends, i.e. if no node can be selected two times in step 2.

Since it must be possible to construct a maximal clique decomposition of any chordal, there must be a maximal clique S that separates (x, y, z) (See 2.15). W.l.o.g let the node u selected in step 2 be in a validating path p_{xy} between x and y and let (u, y, z) be the new asteroidal triple. The selected node u can not be an element of S, since no node connected to a separator of (x, y, z) can form an asteroidal triple with two nodes of the triple. At least one node of p_{xy} must be contained in S. W.l.o.g. let u be in a part of p_{xy} between x and the nearest node to x in p_{xy} that is an element of S. So that in a later iteration x can be selected again, it would have to be in a validating path of (u, y, z). The validating paths from u to y must be a subset of the ones from x to y in (x, y, z), since u lies in p_{xy}. Any validating path of (u, y, z) that crosses x must also be a path between two nodes of (x, y, z) that crosses S. Let p_{xs} be the part of such a path from x to a node in S.

If we assume that there is no such path p_{xs} that does not cross u, then this is trivially true. Assume then that p_{xs} is a path such that $u \notin p_{xs}$. Let see what happens if u is connected to a node in p_{xs}. There are two options:

- If p_{xs} is part of a validating path of (u, y, z) from y to z, then it would intersect the neighborhood of u and which goes against our assumption that it is part of a validating path (u, y, z).
- If p_{xs} is part of a validating path of (u, y, z) from u to z then it would not be a shortest path between u and z that does not intersect the neighborhood of y.

This means if such a validating path of (u, y, z) that contains x exists, then there can be no edge between u and a node in p_{xs}. Since we are assuming u is not connected to any node in p_{xs}, the path p_{xs} and the part of p_{xy} from x to a node in S form a cycle of at least length four. If u is not connected to any chord of the cycle, we know from Lemma 3.9 that its neighbors in the cycle are connected. This would mean that p_{xy} is not an induced path in G. Since this is not possible, and u can not be connected to any other node in p_{xy} for the same reason, it follows that there is a chord between u and a node in p_{xs}, which goes against assumption.

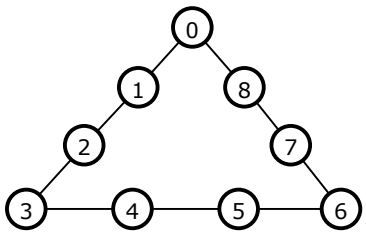

Figure 3.2: Graph with asteroidal triples but no tight asteroidal triple

Since we proved that this configuration is not possible in any case, then the loop must end and the lemma follows. □

Corollary 3.11. *Any chordal graph that does not have a natural path decomposition will contain at least one tight asteroidal triple.*

Expressed in a less formal way, we could say that tight asteroidal triple lie *inside* non-tight asteroidal triples in chordal graphs. The notion of corresponding tight asteroidal triples does not work for general graphs. An example of why corresponding tight asteroidal triples do not make sense in non-chordal graphs can be seen in Figure 3.2. Here $(0, 3, 6)$ is an asteroidal triple. But if we take a node in one of its validating paths and make an asteroidal triple with two other nodes in $(0, 3, 6)$ (e.g. $(\mathbf{1}, 3, 6)$), then the node we removed from the triple is in a validating path of the new asteroidal triple (for $(1, 3, 6)$ the node 0 lies in a validating path from 1 to 6).

Lemma 3.12. *Let (x, y, z) be an non-tight asteroidal triple in a chordal graph G. There exists a tight asteroidal triple (x', y', z') such that every separator of (x', y', z') is also a separator of (x, y, z).*

Proof. If (x, y, z) is a non-tight asteroidal triple it means that there is a node u in a validating path of (x, y, z) that form a tight asteroidal triple with two nodes of this triple. W.l.o.g. let this triple be (u, y, z) and let u lie in a validating path p_{xy} from x to y.

Let S be a separator of (u, y, z). We want to show that S must also be a separator of (x, y, z). Let p_{xu} be the part of p_{xy} that goes from x to u and p_{uy} the part that goes from u to y. The path p_{uy} must be a validating path of (u, y, z), which means that at least one node of p_{uy} is contained in S. The node u can not be contained in S, since no node of an asteroidal triple can be connected to a separator of the triple. Ever validating path p must be an induced path $G[\bigcup p]$, because validating paths are shortest paths. If a node in of p_{xu} would be contained in S, since at least one node of p_{xy} must be contained in S there would be an edge between two non-consecutive nodes in the path p_{xy}. Since then p_{xy} would not be a shortest path it follows that no node of p_{xu} is in S.

If we assume that S is not a separator of (x, y, z), then there must be a path between either x and y or x and z that does not intersect S. It can not be a path between y and z because then S would not be a separator of (u, y, z). W.l.o.g. let there be a path p'_{xy} that goes from x to y that does not intersect S. Then we could reach y from u without intersecting S by using p_{ux} in reverse and then going over p'_{xy}. This would mean that S is not a separator of (u, y, z), which goes against our assumptions. From this follows that all separators of (u, y, z) are separators of (x, y, z).

We can use this argument repeatedly for each new asteroidal triple we get. From the proof of Lemma 3.10 we know that we will repeat this a finite number of times and arrive at a tight asteroidal triple with this process. From this the lemma follows. □

This leads us to or first theorem about the number of separators in any natural decomposition of a chordal graph.

Theorem 3.13. *The minimum number of natural separators needed to separate all asteroidal triples of a chordal graph is the same as the minimum number of separators needed to separate the tight asteroidal triples of a chordal graph.*

Proof. All sets of separators that separator all asteroidal triples in a chordal graph necessarily separate all tight asteroidal triples. From Lemma 3.12 it follows that separating all tight asteroidal triples suffices. □

Theorem 3.13 seems to be a good reason to investigate tight asteroidal triples further. First let us prove the following property of shortest paths in chordal graphs, which help us to characterize validating paths even further.

which we will use to investigate the nature of tight asteroidal triples in chordal graphs.

Lemma 3.14. *Let u and v be two nodes in a chordal graph G. Let $P = (u, p_1, \ldots, p_l, v)$ and $P' = (u, p'_1, \ldots, p'_l, v)$ be two shortest path between u and v where $l \geq 1$ and such that $p_j \neq p'_k$ for $1 \leq j, k \leq l$. Then $\{(p_i, p'_i) \mid 1 \leq i \leq l\} \subseteq E(G)$.*

Proof. Clearly this configuration forms a cycle of length greater then three. The lemma is obviously true for $l = 1$. Let us assume that $l > 1$ then. There can be no chord between two non-consecutive nodes of one of the paths, since this would mean that P and P' are not shortest paths between u and v. There can also not be be a chord in the cycle between two nodes p_i and p'_j where $1 \leq i, j \leq l$ and $|i - j| \geq 2$. If there would be such a chord then there would a shorter path between u and

v over the chord. This means there can only be chords in this cycle between p_i and p'_j iff $|i-j| \leq 1$. We want to show that we have to add all edges where $|i-j| = 0$. It is not possible to make a cycle non-chordal by adding chords to it. This means that we can not block any possible chordalization of the cycle by adding all edges where $|i-j| = 1$. But doing this still leaves every two nodes p_i and p'_j where $|i-j| = 0$ in a cycle of length four. More specifically, let $w \in P$ be a node in one of the paths. Let w be either u or a node in $\{p_1, \ldots, p_{l-1}\}$. The node w must be connected to two nodes p_k and p'_k where $1 \leq k \leq l$. The nodes p_k and p'_k must again be connected to the same node w', which is either v or p_{k+1}. This means that there is a cycle of length four (w, p'_k, w', p_k, w), where (w, p_k, w') and (w, p'_k, w') are two shortest path between w and w'. At the beginning we concluded that there must be an edge between p_k and p'_k in such a configuration. Since we can do this for any pair (p_i, p'_i), $1 \leq i \leq l$, the lemma follows. □

Corollary 3.15. *Let (x, y, z) be a tight asteroidal triple in a chordal graph. All nodes that are in a validating path between two nodes $u, v \in \{x, y, z\}$ at the same distance to u will form a clique.*

Let us investigate separators of tight asteroidal triples by looking at the maximal cliques of the graph that cut some of its validating paths. This simplifies the following arguments as it enables us to work with sets of nodes instead of sets of validating paths.

This means that instead of looking at the paths it will be easier to look at the set of nodes contained in the validating paths.

Definition 3.16 (Validating set)**.** *Let (x, y, z) be an asteroidal triple of a chordal graph G and $P = \{p_1, \ldots, p_k\}$ be the set of all validating paths of (x, y, z). Then we call the set $\{u \in V(G) \mid p \in P, u \in p\}$ the* validating set *of (x, y, z).*

First, let us prove that for any node in a tight asteroidal triple of chordal graph, we can find a maximal clique that contains the node and all the nodes of the validating set of the triple with which the node is connected. This will be the corollary of the next lemma.

Lemma 3.17. *Let (x, y, z) be a tight asteroidal triple in a chordal graph G. Furthermore let M be the validating set of (x, y, z). Then for any node $u \in \{x, y, z\}$ the set $N(u) \cap M$ is a clique.*

Proof. From Lemma 3.14 we know that all nodes in the neighborhood of u that are in a path from u to a node $v \in \{x, y, z\} - \{u\}$ and have the same distance to u must form a clique. This means that the nodes that have distance one to u, which are the ones

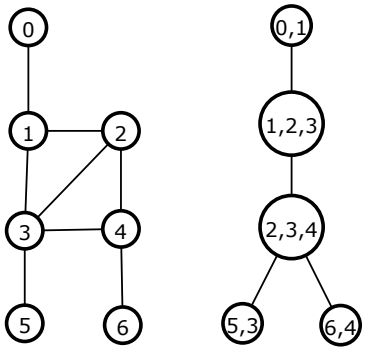

Figure 3.3: A chordal graph with a tight asteroidal triple $(0, 5, 6)$ and a tree-decomposition of the graph with two separator cliques $\{1, 2, 3\}$ and $\{2, 3, 4\}$

that are in $N(u)$, must also form a clique. It is left to show that for two distinct nodes $v, w \in \{x, y, z\} - \{u\}$ the node v' of any validating path p_{uv} from u to v for which $(u, v') \in E(G)$ and the node w' in any validating path p_{uw} from u to w for which $(u, w') \in E(G)$ must be connected by an edge.

In a chordal graph there has to be a maximal clique S that separates (x, y, z). This means that since both p_{uv} and p_{uw} cross the clique S the tree nodes v', u and w' must lie in a cycle. The node u can no be connected to any other nodes of p_{uv} or p_{uw} besides v' and w', else p_{uv} and/or p_{uw} would not be shortest paths. Then from Lemma 3.9 it follows that $(v', w') \in E(G)$. Since this is valid for any such pair of nodes connected to u the lemma follows. □

Corollary 3.18. *For any node in a tight asteroidal triple there is a maximal clique containing the node and all the nodes in the validating set connected to the node.*

We will analyze now the relationship between the maximal cliques which are separators of a tight asteroidal triple and the cliques that are not separators, but intersect such separators.

Definition 3.19 (Separator Clique)**.** *The separator cliques of a tight asteroidal triple (x, y, z) in a chordal graph G are the maximal cliques of G that separate (x, y, z).*

Definition 3.20 (Connection Clique)**.** *Let (x, y, z) be a tight asteroidal triple in a chordal graph G and let $\mathcal{M}$ be the set of its separator cliques. A* connection clique *is a maximal clique $C \notin \mathcal{M}$ of G for which there is a separator clique $S \in \mathcal{M}$ such that $C \cap S \neq \emptyset$.*

Definition 3.21 (Connection Cut)**.** *Let (x, y, z) be a tight asteroidal triple of a chordal graph and let $\mathcal{M}$ be the set of its separator cliques. Furthermore let C be one of its connection cliques. The* connection cut *$c(C)$ of C is an intersection $C \cap S$ with a separator clique $S \in \mathcal{M}$,*

such that there is no separator clique $S' \in \mathcal{M}$ for which $|C \cap S'| > |C \cap S|$.

Definition 3.22 (Maximal Cut)**.** *Let (x, y, z) be a tight asteroidal triple of a chordal graph and let $\mathcal{C}$ be the set of its connection cliques. A connection clique $C \in \mathcal{C}$ has a* maximal cut *if there is no other connection clique $C' \in \mathcal{C}$ such that $c(C) \subsetneq c(C')$.*

A rough sketch of the algorithm we want to create based on these notions is the following:

1. Find a tight asteroidal triple (x, y, z) in the chordal graph G.
2. Find all the separator cliques $\mathcal{M}$ of (x, y, z).
3. Call the algorithm recursively on $G[\bigcup \mathcal{M}]$ which gives us the tree-decomposition $\mathcal{T}_{\mathcal{M}}$.
4. Partition all the maximal cliques of G that are not separator cliques of (x, y, z) in such a way that a tree-decomposition of a cell of the partition can be connected to $\mathcal{T}_{\mathcal{M}}$ with a single edge.
5. Call the algorithm on each cell of the partition and get the trees $\{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$.
6. Make a tree-decomposition of the current graph by connecting each element of $\{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ to $\mathcal{T}_{\mathcal{M}}$ with a single edge.

Furthermore we are going to show that we can make such an algorithm where the number of joins in the resulting tree-decomposition will be bounded by the number of tight asteroidal triple in the chordal graph. The work needed to show that a partition like the one described in step 4 of the previous algorithm sketch is rather involved. We will start by defining certain things that will be useful later.

Definition 3.23 (Connection Partition)**.** *Let (x, y, z) be a tight asteroidal triple of a chordal graph G, let $\mathcal{M}$ be the set of separator cliques of (x, y, z) and let $\mathcal{C}$ be the set of connection cliques of (x, y, z). The connection cuts of G can be ordered as a partially ordered set with respect to set inclusion relation. Let $\mathcal{I}$ be the set of maximal elements in this partially ordered set. The* connection partition *is a partition of $\mathcal{C}$ with respect to the elements of $\mathcal{I}$. A connection clique $C \in \mathcal{C}$ can be in a cell belonging to $I \in \mathcal{I}$ if $c(C) \subseteq I$.*

Notice that a connection partition must not be unique. Before we use this partition as a basis for our algorithm we have to proof certain properties of connections cliques.

Lemma 3.24. *Let (x, y, z) be a tight asteroidal triple of a chordal graph G and let $\mathcal{M}$ be the set of all separator cliques of (x, y, z). If $C \notin \mathcal{M}$ is a maximal clique of G then there is no natural tree-decomposition $\mathcal{T}$ of G, such that C is in a path between two separator cliques $S, S' \in \mathcal{M}$.*

Proof. Since S, C and S' are maximal cliques and they are in a path, it means that there must be a node $u \in S$ such that $u \notin C, S$ and a node $v \in S'$ such that $v \notin A, S$ and $u \neq v$. We know then that C must be a separator of u and v (see Proposition 2.2). C is by construction not a separator of (x, y, z), which means that there is a path p between two nodes in the triple that does not cross C. A node of p must be contained in both S and S', which means that we can use part of p to reach v from u without crossing C. This would mean that C is not a separator of u and v, which is a contradiction. □

Lemma 3.25. *Let (x, y, z) be a tight asteroidal triple in a chordal graph and let C be one of its connection cliques. The connection cut of C is unique.*

Proof. If the connection cut of C would not be unique there would have to be two separator cliques S and S' of (x, y, z) such that $|C \cap S| = |C \cap S'|$ but $C \cap S \neq C \cap S'$. This means that one of two things must be true:

- C must lie in a path between S and S'. The previous lemma makes this impossible, since C is by definition not a separator.
- There must be a separator clique S'' such that $C \cap S \subsetneq C \cap S'' \supsetneq C \cap S'$. This would mean that neither $C \cap S$ nor $C \cap S'$ are connection cuts of C, since the intersection $C \cap S''$ would be bigger than both of them. This would also go against our assumptions.

Since both options are impossible, the lemma follows. □

Lemma 3.26. *Let (x, y, z) be a tight asteroidal triple in a chordal graph G and let C and C' be two connection cliques of (x, y, z). If $c(C) \not\subseteq c(C')$ and $c(C') \not\subseteq c(C)$ then there is no node $u \notin c(C), c(C')$ such that $u \in C \cap C'$.*

Proof. Let $\mathcal{T}$ be natural tree-decomposition of G. Let us assume that a node u exists. For both C and C' there must be a path in $\mathcal{T}$ that connects them to the nearest separator cliques in $\mathcal{T}$ of (x, y, z), $S \supseteq c(C)$ and $S' \supseteq c(C')$ respectively. Since neither $c(C) \subseteq c(C')$ nor $c(C') \subseteq c(C)$, it follows that neither $S \cap C \not\subseteq C'$ nor $S' \cap C' \not\subseteq C$. This means that C can not lie in a path from

C' to S' nor can C' lie in a path from C to S. From Lemma 3.24 we know that neither C nor C' can lie in a path of $\mathcal{T}$ between to separator cliques of (x, y, z). This means that there must be a path in $\mathcal{T}$ from C to C' such that S and S' are in the path. Since both C and C' contain u, and then every bag in the path between them must also contain u, it follows that $u \in S, S'$. But this would mean that $u \in c(C), c(C')$, which goes against our assumptions. □

Corollary 3.27. *Let (x, y, z) be a tight asteroidal triple of a chordal graph, let $\mathcal{M}$ be the set of separator cliques of (x, y, z) and let C be a connection clique of (x, y, z) with a maximal cut. For any node $u \in C - c(C)$, $N(u) \cap \bigcup \mathcal{M} = c(C)$.*

Proof. If C has a maximal cut and u is connected to some node in a separator clique that is not in $c(C)$, then there must be some connection clique C' that contains part of these edges. The connection cut $c(C')$ can then not be a subset of $c(C)$. Since C is a maximal cut it can not be a proper superset either. This means that if C and C' both contain u, it would go against Lemma 3.26. □

Lemma 3.28. *Let (x, y, z) be a tight asteroidal triple in a chordal graph G, let $\mathcal{M}$ be the set of separator cliques and $\mathcal{C}$ be the set of connection cliques of (x, y, z). Furthermore let H be a connected component of $G[V(G) - \bigcup \mathcal{C}]$. Let also C and C' be two connection cliques with maximal cuts such that $c(C) \nsubseteq c(C')$ and $c(C') \nsubseteq c(C)$. Then either $G[\bigcup \mathcal{M} + V(H) + C]$ or $G[\bigcup \mathcal{M} + V(H) + C']$ has more than one component.*

Proof. Notice the graph $G[\bigcup \mathcal{M} + V(H)]$ has to have two components. $G[\bigcup \mathcal{M}]$ can only have one component because else there would have to be a maximal clique which would be a separator of two components of $G[\bigcup \mathcal{M}]$, which would go against Lemma 3.24. $G[V(H)]$ has one components by definition. There can be no edge between a node of $G[V(H)]$ and $G[\bigcup \mathcal{M}]$ because any such node must be contained in the part of a connection clique that is not its connection cut. None of these nodes are present in $G[\bigcup \mathcal{M} + V(H)]$. Let us assume that $G[\bigcup \mathcal{M} + V(H) + C]$ and $G[\bigcup \mathcal{M} + V(H) + C']$ have both just one component. We are going to show that this would mean that G is not chordal.

Let C and C' be two connection cliques with maximal cuts. Let $u \in C - C'$ and $v \in C' - C$ be two nodes such that neither of them is in $c(C)$ nor in $c(C')$. We know from Lemma 3.26 that such nodes must exist. Furthermore let $u' \in c(C) - c(C')$ and $v' \in c(C') - c(C)$. What we are going to show is that in this configuration there are node u, u', v and v' that lie in a chordless cycle of at least length four.

Now we are going to show that the nodes u', v' either form an edge $(u', v') \in E(G)$ or there is a shortest path $p_{u'v'}$ from u' to v' that does not intersect $C \cap C'$. The node u' is in $c(C)$, which means that it is contained in a separator clique S of (x, y, z). There must be a path p_{xyz} between two nodes of (x, y, z) that does not intersect C, else C would be a separator of (x, y, z) instead of a connection clique. Since there must be a node of p_{xyz} in S, u must be connected to a node in p_{xyz}. The node v' must also be contained in some separator clique S' and there must be a node of p_{xyz} in S. Now there are two options:

- The node v' is a node in p_{xyz}. Then we can reach v' from u' over the path p_{xyz} without intersecting $C \cap C'$.
- The node v' is not a node in p_{xyz}. Let w be the node of p_{xyz} to which v' is connected. Then we can reach v' from u' using the part of p_{xyz} that goes to w and then to v' without intersecting $C \cap C'$.

There must be a shortest path p_{uv} from u to v that only contains node of $V(H)$. Let us assume that there is a path p_{uv} where if a node of p_{uv} is connected to nodes in $\bigcup \mathcal{M}$ it must be connected to a subset of $C \cap C'$. Then we can show that there is a chordless cycle of at least length four in G:

Let us look that the cycle $(u, u', p_{u'v'}, v', v, p_{uv}^R, u)$. In any case this cycle contains a chordless cycle of at least length four:

- From Corollary 3.27 we know that u and v can not be connected to any node to which they are note connected in C and C' respectively. This means that there is no chord $(u, v') \in E(G)$, $(v, u') \in E(G)$ nor between u or v and a node of $p_{u'v'}$.
- $p_{u'v'}$ and p_{uv} are shortest paths, which means that there is no edge between two non-consecutive nodes.
- No node of $p_{u'v'}$ is connected to a node of p_{uv}, since no node in $p_{u'v'}$ is in $C \cap C'$ and node in p_{uv} can only be connected to node in $C \cap C'$.
- Neither v' nor u' are in $C \cap C'$ so no node in p_{uv} can be connected to either of them.

These are all possible chords. If $p_{u'v'}$ and p_{uv} are empty, i.e. $(u, v) \in E(G)$ and $(u', v') \in E(G)$, then there is a chordless cycle of length four. That means that if a path p_{uv} exists we would have proofed the lemma.

Let p_{uv} be a shortest path between u and v that contains a node w such that $N(w) \cap \bigcup \mathcal{M} \not\subseteq C \cap C'$. From Corollary 3.27

we know that there must be a connection clique with a maximal cut C'' that contains all these edges. The connection cut $c(C'')$ can not be a subset of $c(C)$ and $c(C')$. It can not be a proper superset of $c(C)$ or $c(C')$ either, since then C and C' would then not have maximal cuts. This means that either $c(C) \nsubseteq c(C'')$ and $c(C'') \nsubseteq c(C)$ or $c(C') \nsubseteq c(C'')$ and $c(C'') \nsubseteq c(C')$. W.l.o.g. let us say that $c(C) \nsubseteq c(C'')$ and $c(C'') \nsubseteq c(C)$. This means that we can apply the logic of this proof to C and C'' from the beginning. This time we can take $w \in C''$ instead of $u \in C'$ as the node that is not in the connection cut of C''. As a shortest path between u and w we can take a part of p_{uv}. Since p_{uv} is always a shortest path, this can only happen a finite number of times.

As such we will reach a configuration were we can start the proof with two connection cliques C and C' where there is a path p_{uv} between u and v where no node is connected to a node in $\bigcup \mathcal{M}$ that is not in $C \cap C'$. As we showed before, then we can find a chordless cycle of at least length four, which goes against the chordality of G. □

Definition 3.29 (Clique Components Partition)**.** *Let (x, y, z) be a tight asteroidal triple of a chordal graph G, let $\mathcal{M}$ be the set of separator cliques and $\mathcal{C}$ the set of connection cliques of (x, y, z) and let $\mathcal{R}$ be the set of maximal cliques of G that are neither separator nor connection cliques of (x, y, z). Furthermore let CP be a connection partition of $\mathcal{C}$. Then a* clique component partition *is a partition of $\mathcal{C} + \mathcal{R}$ which is constructed as follows:*

- *Start with CP which already partitions $\mathcal{C}$. We will keep the same number of cells as CP has.*
- *From Lemma 3.28 we know that for a connected component H of $G[V(G) - \bigcup \mathcal{M}]$ all the edges that connect a node in H to a node in $G[\mathcal{M}]$ must be in connections cliques which are all contained in a single cell of CP.*
- *Every maximal clique $R \in \mathcal{R}$ must be contained in a component H of $G[V(G) - \bigcup \mathcal{M}]$.*
- *The maximal clique R is then contained in the cell that contains all connection cliques $C \in \mathcal{C}$ such that $G[\mathcal{M} + V(H) + C]$ has a single component.*

Lemma 3.30. *Let (x, y, z) be a tight asteroidal triple of the chordal graph G and let $\mathcal{M}$ be the set of its separator cliques. Furthermore let $CCP = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ be a clique components partition of (x, y, z). We can then construct a natural tree-decomposition of G by constructing one natural tree-decompositions of $\mathcal{M}$ and and a natural tree-decomposition for each cell in CCP and then connecting each of these with one edge to a tree-decomposition decomposition of $\mathcal{M}$.*

Proof. From Lemma 3.24 we know that the maximal cliques in $\mathcal{M}$ must form a single component in any strict maximal clique decomposition of G. We can therefore first construct a tree-decomposition of bags containing the cliques in $\mathcal{M}$ and use it in a tree-decomposition of the whole of G.

Let $\mathcal{C} \in CPP$ be a component of the clique components partition of (x, y, z). The subgraph $G[\bigcup \mathcal{C}]$ must be by construction a single component in the graph. Also by construction, there is a set S that is a superset of every connection clique contained in $\mathcal{C}$. That means that we can connect a bag containing a clique of $S \subseteq M \in \mathcal{M}$ and a a bag containing a connection clique $S \subseteq C \in \mathcal{C}$ and get a valid tree-decomposition of $G[\bigcup \mathcal{M} + \bigcup \mathcal{C}]$.

The subgraph $G[\bigcup \mathcal{M}]$ separates all subgraphs $G[\bigcup \mathcal{C}]$, this means that in no tree-decomposition of G a bag containing a clique in one component can be directly connected to a bag containing a clique of another component. This proofs the lemma. □

This lemma gives us a footing to develop a recursive algorithm that works on the maximal cliques of a chordal graph, which finds a tight asteroidal triple and its separator cliques, calculates a clique component partition and then is called recursively on each cell $\mathcal{C}$ of the partition and the separator cliques, until no tight asteroidal triple can be found anymore. If we can not find a tight asteroidal triple we know from Corollary 3.11 that there is a path decomposition of $\mathcal{C}$. At that point we can end the recursion.

But before we describe this algorithm in detail let us find the boundaries of how many joins a tree-decomposition constructed this way could have with respect to the number of tight asteroidal triples in the graph.

First a preliminary result we will need to locate asteroidal triples between connections cliques:

Lemma 3.31. *Let (x, y, z) be a tight asteroidal triple in a chordal graph, let $\mathcal{M}$ be the set of separator cliques of (x, y, z) and let C be connection clique of (x, y, z) such that there is no $u' \in \{x, y, z\}$ for which $c(C) \subseteq N(u')$. Then there are two nodes $u, v \in \{x, y, z\} + \bigcup \mathcal{M}$ and a node $w \in C - c(C)$ such that (u, v, w) is a tight asteroidal triple.*

Proof. If C is a connection clique of (x, y, z) it can not be separator of (x, y, z), so there must be two nodes in (x, y, z) that are not separated by C. Let $u, v \in \{x, y, z\}$ be two nodes not separated by C. We want show that (u, v, w) is a tight asteroidal triple. We know there is a path from u to v that does not intersect C, which means that it does not intersect $N(w)$ either. We still have to

show that there is a path from u to w that does not intersect $N(v)$ and a path from v to w that does not intersect $N(u)$.

By assumption there must be a node $w' \in c(C)$ that is not in the neighborhood of u, else $c(C)$ would be a subset of $N(u)$. There must also be a path $p_{z'v}$ from the third node in the triple $z' \in \{x, y, z\} - \{u, v\}$ to v that does not intersect $N(u)$. Either w' is a node in $p_{z'v}$ or, since a node of $p_{z'v}$ must be contained together with w' in a separator clique S, w' must be connected to a node in $p_{z'v}$, which means there is a path from w to v over w' that does not cross the neighborhood of u. The same argumentation can be done to show that is a path from w to u that does not cross the neighborhood of v. It follows that (u, v, w) is an asteroidal triple.

Let p be a validating path of (u, v, w) from w to some other $u' \in \{u, v\}$. The node $w' \in p$ connected to w must be in $c(C)$. Since all nodes in $c(C)$ must be contained in some separator of (x, y, z) it means that w' must be contained in a separator clique of (x, y, z) and thus it is connected to a separator of u and v. This means that (u, v, w') can not be an asteroidal triple. From this is follows that if (u', v', z') is the correspind tight asteroidal triple of (u, v, w') then $w \in \{u', v', z'\}$. The other two nodes of must lie in the path to a separator clique S of (u', v', z'), which is also a separator clique of (x, y, z). This means that these two nodes are either u or v or a node in a separator clique. □

Corollary 3.32. *Let (x, y, z) be a tight asteroidal triple in a chordal graph, let $\mathcal{M}$ be the set of separator cliques of (x, y, z) and let $CPP = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ be a clique components partition of (x, y, z). Then there is for every cell $\mathcal{C} \in CPP$ in the partition there is a node $u \in \bigcup \mathcal{C}$ that forms a tight asteroidal triple (u, v, w) with two nodes $u, v \in \{x, y, z\} + \bigcup \mathcal{M}$.*

Proof. If $u \in \{x, y, z\}$ then this is trivial. Let C be a connection clique such that $c(C) \subseteq N(u')$ where $u' \in \{x, y, z\}$. This means that $\{u'\} + c(C)$ forms a clique, which means that there must be some connection clique $C' \supseteq \{u'\} + c(C)$, which means that any such connection clique C must be contained in a cell that also contains C' and thus also contains a node of (x, y, z). Every cell must contain at least one connection clique. This means that every cell which does not contain a node of (x, y, z) must contain a connection clique that because of Lemma 3.31 must contain a node that makes a tight asteroidal triple (u, v, w) with two nodes $u, v \in \{x, y, z\} + \bigcup \mathcal{M}$. It also follows that there must be $k - 2$ such tight asteroidal triples. □

We know for now that we are able for a tight asteroidal triple to find all the separator cliques and make a tree out of them. We also must be able to take all the cliques that contain the cut of

one node in the triple with its validating set and connect it to bags in the validating set and connect them to this path. With the added knowledge of the existence of tigh asteroidal triples that Lemma 3.31 gives us, let us try to pinpoint the number of joins necessary.

Lemma 3.33. *Let (x, y, z) be a tight asteroidal triple of the chordal graph G, let $\mathcal{M}$ be the set of separator cliques of (x, y, z) and let $CCP = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ be a clique components partition of (x, y, z). The minimum number of joins necessary in any natural tree-decomposition of G to separate all asteroidal triples (x', y', z') such that $x' \in \bigcup \mathcal{C} \in CCP$ and $y', z', \in \{x, y, z\} + \bigcup \mathcal{M}$ is $k - 2$.*

Proof. For every asteroidal triple in G there must be a separator S contained in some bag in any tree-decomposition of G (see Lemma 3.4). Let (x', y', z') and (x'', y'', z'') be two asteroidal triples such that at least two nodes $u \in \{x', y', z'\}$ and $v \in \{x'', y'', z''\}$ are contained in different cells of CCP. For them to share a a join there would have to be two connection cliques $u \in C$ and $v \in C'$ that lie in a path to S. In Lemma 3.30 we showed that this is not possible.

Since we know from Corollary 3.32 that there have to be at least $k - 2$ such tight asteroidal triples, it follows that there will be at least $k - 2$ joins in any natural tree-decomposition of G. □

Lemma 3.34. *Let (x, y, z) be a tight asteroidal triple of the chordal graph G and let $CCP = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ be a clique components partition of (x, y, z). The maximum number of joins necessary in a tree-decomposition of G to separate all tight asteroidal triples (x', y', z') contained in three distinct cells $\mathcal{C}, \mathcal{C}', \mathcal{C}'' \in CCP$ of the partition $[x' \in \bigcup \mathcal{C}$, $y' \in \bigcup \mathcal{C}'$ and $z' \in \bigcup \mathcal{C}'']$ is $2k$.*

Proof. From Lemma 3.30 we know that we can create a tree-decomposition of the cliques of each cell in the clique components partition and then connect it with one edge to a path decomposition of the separator cliques. Each such edge can only create two more joins. In such a tree-decomposition every (x', y', z') is separated. □

We can now construct Algorithm 3.1 based on the notions we have been developing.

Lemma 3.35. *Algorithm 3.1 calculates a natural tree-decomposition of a chordal G where if t is the number of tight asteroidal triples in G the number of joins in the tree-decomposition is less or equal to $6t$.*

Proof. In every call of the recursive algorithm we create at most $2k$ joins, where k is the number of cells in the clique components partition. From Lemma 3.31 we know that the number of tight

asteroidal triples we are separating at this point must be at least $k-2$. Let $\mathcal{M}$ be the set of separator cliques of a tight asteroidal triple (x, y, z) selected in line 6. Furthermore let $\{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ be the tree-decompositions from the recursive calls on line 13 on each cell of the clique component partition of (x, y, z). The minimum number of joins we may have to add at this point is zero. This is the case if there are already k leaves in the tree-decomposition $\mathcal{T}_\mathcal{M}$ of the graph that only contains separator cliques $G[\mathcal{M}]$ (line 8) and we can connect a leaf of each $\mathcal{T} \in \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ to a leaf of $\mathcal{T}_\mathcal{M}$. The worst case is clearly when we have to add zero joins but we add $2k$ by looping over line 15. Let us show that nevertheless we can bound the number of joins on t.

In the final call of the recursion there can be no tight asteroid triples in the graph induced on the nodes of the separator cliques. This means the separator cliques will become a path. Every of the $k-2$ joins necessary will must be created when connecting each element of $\{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ with $\mathcal{T}_\mathcal{M}$. This will create at most $2k$ joins. The ration between the minimum number of joins we have to create and the worst case is $2k/k-2$. Three is the worst case of $2k/(k-2)$, since with growing k the fraction $2k/(k-2)$ goes against two and k will be at least three. This means that here we create at most six times to many joins.

In every other call of the recursion in lines 8 or 13 the graph on which we call the function recursively has to have at least one tight asteroidal triple less than G. This means that at every step, $k-2$ already existing bags may suffice, but the algorithm creates $2k$ on top of it. The ratio between the minimum and the number of joins we create on top of it is again $2k/(k-2)$. This can happen at most t times since t bounds the recursion depth. The number of cells in a clique components partition has to be at least three. This means that in every recursive call we create 6 times more joins that necessary, which gives us the desired bound of $6t$. □

From this lemma and since we can always apply Algorithm 3.1 to any chordal graph the following theorem must be true.

Theorem 3.36. *If a chordal graph has t tight asteroidal triples, there exists a natural tree-decomposition of the graph with no more than $6t$ joins.*

We want to show now that Algorithm 3.1 is a polynomial time $6t$-approximation algorithm. First we will need to proof that some of its steps can be done in polynomial time.

Lemma 3.37. *Let G be a chordal graph. All tight asteroidal triples in G can be found in polynomial time.*

Algorithm 3.1: MINIMUMNUMBERJOIN

input : A chordal graph G.
output: A tree-decomposition of G with a minimum number of joins.

```
1  C = All maximal cliques in G
2  TAT = All tight asteroidal triples in G
3  if TAT = ∅ then
4  |  return makePath(C)
5  end
6  (x, y, z) ∈ TAT
7  M = Separator cliques of (x, y, z)
8  T_S = minNumJoins(G[⋃ M])
9  C = C − M
10 {C_1, ..., C_k} = cliqueComponentPartition(G, C, M)
11 RES = T_s
12 foreach G' ∈ {G[⋃ C_1], ..., G[⋃ C_k]} do
13 |  T = minNumJoins(G')
14 |  C = (∪T) ∩ (∪P)
15 |  connectTwoBags(RES, T)
16 end
17 return RES
```

Proof. First notice that the number of triples of nodes in a graph is bounded by n^3 if $n = |V(G)|$. Thus if each one of them can be tested to be a tight asteroidal triple in polynomial time we are done.

Let us show that we can test if a triple of nodes (x, y, z) in G is an asteroidal triple in polynomial time. If the nodes in the triple are not pairwise distinct then it can not be an asteroidal triple. Neither can it be an asteroidal triple if for $u, v \in \{x, y, z\}$ the edge $(u, v) \in E(G)$ exists.

Let us then assume that the three nodes in (x, y, z) are pairwise distinct and form an independent set. Let $u, v \in \{x, y, z\}$ be two distinct nodes in the triple and let $w \in \{x, y, z\} - \{u, v\}$. If u and v are in the same component of $G[V(G) - N[w]]$ then there is a path in G from u to v that does not intersect the neighborhood of the third node in the triple. We can clearly test this in polynomial time for each pair $u, v \in \{x, y, z\}$. Iff a path in $G[V(G) - N[w]]$ between u and v exists for every pair $u, v \in \{x, y, z\}$ then (x, y, z) is an asteroidal triple. This means that we can test in polynomial time if a triple (x, y, z) is an asteroidal triple.

We want to use this to show that we can test in polynomial time if a triple (x, y, z) is a tight asteroidal triples. First we can test in polynomial time if it is an asteroidal triple. If it is not, it can not be a tight asteroidal triple either. If it is, then for every pair $u, v \in \{x, y, z\}$ we make the following test: Starting at u we calculate a breadth first search tree in G. From this tree we can read all shortest path from u to v. For every node w' in such a path, we can test for every pair of node in $u', v' \in \{x, y, z\}$ if (u', v', w') is an asteroidal triple in polynomial time. Only if every such test fails for every w',u',v',u and v, then (x, y, z) is a tight asteroidal triple. Since this is a polynomial number of tests, each one of the them taking polynomial time, the lemma follows. □

Surely the algorithm presented in the proof of Lemma 3.37 is rather *naïve* and its running time could be improved greatly in practice.

Lemma 3.38 ([26])**.** *A path-decomposition of an circular-arc graph can be found in polynomial time.*

Corollary 3.39. *A path-decomposition of an interval graph can be found in polynomial time.*

Proof. All interval graphs are circular-arc graphs. □

Theorem 3.40. *There is a polynomial time 6t-approximation algorithm for the minimum join number problem on chordal graphs G, where t is the number of tight asteroidal triples in G.*

Proof. From the definition of a clique components partition we can see that a clique components partition can be found in polynomial time. The depth of the recursion is bounded by the number of tight asteroidal triples which is itself bounded by n^3, where $n = |V(G)|$. This together with Lemma 3.37 and Corollary 3.39 means that Algorithm 3.1 runs in polynomial time. We also know that Algorithm 3.1 calculates a $6t$-approximation. From this the theorem follows. □

There seems to be a strong connection between the number of tight asteroidal triples in a chordal graph and the number of joins necessary in any natural tree-decomposition of such a graph. That is why we conjecture that the following holds true.

Conjecture 3.1. *The necessary number of joins in any natural tree-decomposition of a chordal graph G is the same as the number of separators necessary to separate all tight asteroidal triples in G.*

It seems that we could make an even stronger conjecture.

Conjecture 3.2. *Algorithm 3.1 has a run that generates a tree-decomposition with a minimum number of joins.*

The problem of finding the natural tree-decomposition of a chordal graph with a minimum number of joins remains nevertheless open.

Conjecture 3.3. *The problem of deciding the minimum number of joins in a natural tree-decomposition of a chordal graph is* NP*-complete.*

3.1.1 *Heuristical Improvements of Algorithm 3.1*

Some of the places where improvements could be made in a practical implementation would be:

- Take the bags we are connecting in line 15 and try to move them so that they become leafs. Doing this we try to minimize the number of joins we created when connecting several partial tree-decompositions into one.
- When calculating a clique component partition, try to connect components over a connection clique where all bags in one component are a subset of one of the connection cuts. Only if this is the case two connection cliques that contain nodes of different tight asteroidal triples as of Lemma 3.31 can all be in a path to a separator clique, and thus it increases the chances that we can find a leaf to which we can connect a separator clique.
- When selecting a tight asteroidal triple in line 6 select one that has a single separator clique if there is one. If we can find such a tight asteroidal triple, we know it must be separated by this clique, as does every connection clique that contains a node of a tight asteroidal triple as of Lemma 3.31.
- If there is no tight asteroidal triple with a single separator clique, try to find a tight asteroidal triple such that the induced graph over the separator cliques has a path decomposition. In this case, if we have k components, we know that we will have to create at least $k - 2$ joins when connecting the tree-decompositions of the components to the separator clique path. This lowers the relative number of joins the algorithm can create unnecessarily.
- If no such tight asteroidal triples exist, take one with a minimum number of tight asteroidal triples itself. This increases the chances that the tree made of separator cliques

will have a small number of leafs, and thus, again, this lowers the relative number of joins the algorithm can create unnecessarily.

3.2 SIZE OF JOIN SEPARATORS

We have investigated the number of joins necessary in a natural tree-decomposition of a chordal graph, but would still like to say something about the size of the join bags. We are going to make some assumptions here about the strict maximal clique decompositions that we can construct, but the results will be applicable as a heuristic.

Definition 3.41 (Minimum Separator of an Asteroidal Triple)**.** *A* minimum separator of an asteroidal triple *is a separator of an asteroidal triple S, where no strict subset $S' \subsetneq S$ is also a separator of the same triple.*

When it came to the number of joins in a tree-decomposition we only looked at natural tree-decompositions. But if we look at the definition of minimum separators for asteroidal triples, the question if we should use a separator that is not a clique in the chordal graph arises. Maybe it would be better if we construct a tree-decomposition where all the bags would contain maximal cliques except the join bags, which could contain anything, more specifically, sets that are not necessarily cliques in the graph. We will see that this is not the case, but first we need to proof some things about what join bags have to contain in any possible tree-decomposition.

Lemma 3.42. *Let C_1, C_2 and C_3 be three different cliques. Then $S = C_1 \cap C_2 + C_1 \cap C_3 + C_2 \cap C_3$ is also a clique.*

Proof. Let us take two nodes $u, v \in S$ where $u \neq v$. If u and v are in the same clique then there is an edge between them. Since they are both in S it means that they where contained in one of the three possible cuts between the cliques. If they are contained in the same cut, they have to be connected since there would be two cliques that contain both. Any of the cuts between two cliques share a clique, which means that even if they are contained in S because they are contained in two different cuts, these cuts will have a clique in common, which means that there is a clique that contains both nodes. Since these are all possible cases and this is true for all pairs in S, the lemma follows. □

Lemma 3.43. *The optimal natural tree-decomposition of a chordal graph and the optimal non-strict maximal clique tree-decomposition of a chordal is the same.*

Proof. Because of the properties that a tree-decomposition has to have, every join bag has to contain any node that appears in at a least two of the bags connected to the join bag. That together with Lemma 3.42 means that the content of any join bag will contain a clique in the graph and that this clique is sufficient. That means that if we have a valid tree-decomposition of a chordal graph where all introduce bags are maximal cliques and join bags contain any set of nodes of the graph, we can reduce the join bags until they only contain cliques. From this the lemma follows. □

Theorem 3.44. *Let* $s = \sum_{X \in \mathcal{J}(\mathcal{T})}(d(X) - 2) \cdot |X|$ *be the smallest sum for any natural tree-decomposition* $\mathcal{T}$ *of a chordal graph* G *and let* $s' = \sum_{X \in \mathcal{J}(\mathcal{T}')}(d(X) - 2) \cdot |X|$ *for any tree-decomposition of* G*. If* $s' < s$ *then the width of* $\mathcal{T}'$ *is bigger than the width of* $\mathcal{T}$*.*

Proof. This follows from Lemma 3.43 □

This means that if we want to make the join bags of a tree-decomposition of a chordal graph smaller than in the optimal natural tree-decomposition, we will have to incur the penalty of increasing the width.

The problem of minimizing the sum over the sizes of the joins of a natural tree-decomposition seems to be a lot harder than finding the minimum number of joins. It is not even clear if the natural tree-decomposition $\mathcal{T}$ where $\sum_{X \in \mathcal{J}(\mathcal{T})}(d(X) - 2) \cdot |X|$ is minimum has the same number of joins as the maximal clique decomposition with a minimum number of joins. Nevertheless it seems natural to construct a heuristic to find good natural tree-decompositions by trying to minimize the number of joins and then trying to minimize their size on a chordal graph. This can be useful together with an algorithm to calculate or approximate tree-decompositions of minimum width. The heuristic would then be used in a second step to improve the result of such an algorithm, which would hopefully lower the running time of a normal tree-decomposition algorithm on that instance. We are going to use Remark 3.1 to construct a heuristic that tries to minimize the number and size of the join bags in a tree-decompositions.

Notice that the mapping we use to map all non-strict maximal clique decomposition to strict maximal clique decompositions can be reversed (although the order in which we reverse the steps can change the result). To reverse this mapping we need the following operation:

Definition 3.45 (Join Minimization)**.** *Take a join bag* J *and three bags* B_1*,* B_2 *and* B_3 *that are connected to* J *of a tree-decomposition. Create three new possible join bags taking to bags* $B, B' \in \{B_1, B_2, B_3\}$

Algorithm 3.2: MINIMIZEJOINOPERATION

input : A natural tree-decomposition $\mathcal{T}$ of a chordal graph G, a join bag $J \in \mathcal{B}(\mathcal{T})$ and three children $B_1, B_2, B_3 \in \mathcal{B}(\mathcal{T})$ of J.
output: A tree with a join that separates B_1, B_2 and B_3.

$C_1 = B_1 \cap B_2 + B_1 \cap J + B_2 \cap J$
$C_2 = B_1 \cap B_3 + B_1 \cap J + B_3 \cap J$
$C_3 = B_2 \cap B_3 + B_2 \cap J + B_3 \cap J$
if $|C_1| \leq |C_2|, |C_3|$ **then**
    Disconnect J from B_1 and B_2
    Connect C_1 to J, B_1 and B_2
    return *Changed tree* $\mathcal{T}$
end
if $|C_2| \leq |C_1|, |C_3|$ **then**
    Disconnect J from B_1 and B_3
    Connect C_2 to J, B_1 and B_3
    return *Changed tree* $\mathcal{T}$
end
if $|C_3| \leq |C_1|, |C_2|$ **then**
    Disconnect J from B_2 and B_3
    Connect C_3 to J, B_2 and B_3
    return *Changed tree* $\mathcal{T}$
end

and creating the the possible cliques $C = B \cap B' + B \cap J + B' \cap J$. Take the smallest one and change the tree-decomposition by using C as a join bag to separate J, B and B' from each other, thus decreasing the degree of J. A more formal description is given in Algorithm 3.2.

It is easy to see how we can go back to the tree-decomposition we had before, by changing the new bag containing by a bag containing J and then contracting the edge between the new bag and J, as we would have done in the proof of Lemma 3.5. It is also clear that by using this operation several times on the result of the last join minimization operation we can reach any tree-decomposition that can be converted into the strict maximal clique decomposition we would use the heuristic on. Recursively using the join minimization on a graph, can only decrease the size of the joins and never increase the number of them. This is a good argument as to why it would make a good heuristic. From that moment on, no join minimization operation could decrease the size of the join bags in the tree-decomposition. This heuristic is described formally in Algorithm 3.3. This heuristic

Algorithm 3.3: MINIMIZEJOINSIZEHEURISTIC

input : A strict maximal clique decomposition $\mathcal{T}$ of a chordal graph G.
output: A non-strict maximal clique decomposition of G.

1 $\mathcal{J} =$ All the join bags of $\mathcal{T}$
2 **while** $\exists J \in \mathcal{J}(\mathcal{T})$ *that is not the result of a minimize join operation* **do**
3 $\quad B_1, B_2, B_3 =$ three children of J
4 $\quad \mathcal{T} =$ minimizeJoinOperation($\mathcal{T}$,J,B_1,B_2,B_3)
5 **end**

could be tweaked further by selecting a join in line 2 and its three children in line 3 in an intelligent way.

We have been talking about improving a tree-decomposition heuristically by first using Algorithm 3.1 and then attempting to improve this tree-decomposition using Algorithm 3.3. Doing it this way, would mean that the operations of one algorithm would be completely uncoupled from the other, but clearly which bags we choose to make join bags (we have a degree of freedom on this) in Algorithm 3.1 has an influence on the heuristic. Another way to heuristically improve this is to change the call on line 15 of Algorithm 3.1. If instead of taking any node of the tree of separator cliques we connect the one where the minimize join operator would find the smallest join, it could increase the chances of the later heuristic finding smaller join bags. Intuitively the heuristic would then have a higher probability of improving the size of the joins.

3.2.1 *Idea for a Simple Heuristic to Decrease Join Size in tree-decompositions at the Cost of Increasing Width*

Something that would be useful in practice would be to have an algorithm that tries to find tree-decompositions for general graphs with minimum width but so that all join bags are smaller than some maximum size m. To find such an algorithm seems to be very complicated. A different approach would be to use a two step heuristic:

1. Use one of known algorithms that optimizes treewidth.
2. Operate on the resulting tree-decomposition to get smaller joins.

A similar approach is already used for treewidth. First a tree-decomposition with some known algorithm is found and is

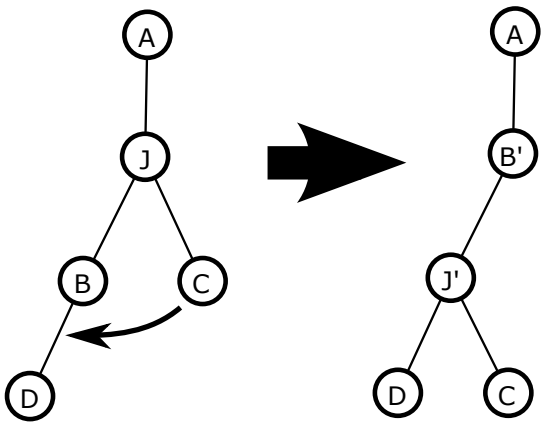

Figure 3.4: Size minimization operation. $J = A \cap B + A \cap C + B \cap C$, $J' = B \cap C + B \cap D + C \cap D$ and $B' = B + J$

then manipulated heuristically trying to make the width even smaller [28]. A special case of this approach is when we want to not have any joins left after the second step. Then we would be calculating the pathwidth of a chordal graph. In [19] it was shown that this problem is NP-hard, which means that even this simplified two step approach is NP-hard.

We propose the operation seen in Figure 3.4 to heuristically improve a given tree-decomposition. The hope is that we can find either a join bag J' at some point by repeating this operation or that we will reach a point where B is a leaf.

There is a reason related to tight asteroidal triples why this may be a good idea. Let us assume that B contains a node x which is part of a tight asteroidal triple (x, y, z) and that J is a separator (x, y, z). After the operation we connect x to all nodes in J. This means that (x, y, z) can not be an asteroidal triple in the underlying chordalization anymore.

This operation could be reapeated recursively looking to look for a small enough join bag, as long as we do not make some introduce bag too big in the process. The problem with this approach would be that on the way to a smaller join we can create an unacceptable big one. But there is no reason why we should constrain ourselves to making a child of a join a child of a different child of the join. We could just look for a place where the join would be small enough, and then add the separator on every bag on the way.

4

CONCLUSION AND FUTURE WORK

4.1 RESULTS IN THIS THESIS

Most of the results in this thesis were based on the notion of tight asteroidal triples and work on graphs that have natural tree-decompositions, which are tree-decompositions where the content of each bag forms a clique in the graph. The class of graphs that have natural tree-decompositions is precisely the class of chordal graphs. We showed that tight asteroidal triples can be found in any chordal graph that does not have a natural path decomposition. We defined a separator of a tight asteroidal triple as a clique in the graph that separates each pair in the triple. There have been two main result, both of them based on this structure and their separators:

- If G is a chordal graph and k is the minimum number of maximal cliques of G needed to separate all tight asteroidal triples in G, then any natural tree-decomposition of G has at least k joins.
- If G is a chordal graph and t is the number of tight asteroidal triples in G, then there is a a polynomial time $6t$-approximation algorithm for the problem of finding a natural tree-decomposition of G with a minimum number of joins. We did not only show that such an algorithm exists, but gave one that may be useful in practice.

Thus we found a lower and an upper bound for the number of joins in a natural tree-decomposition of a chordal graph. There are two reasons why this results are of interest.

First, there is a strong connection between tree-decompositions of general graphs and chordalizations of graphs. Every tree-decomposition of a graph G gives a chordalization of G. Even more, every minimum tree-decomposition of G, measured in treewidth, gives a minimum chordalization of G, measured in number of edges added. This means that the previous result result on natural tree-decomposition for chordal graphs tells us something about general tree-decompositions of general graphs:

Let G be any graph, $\mathcal{T}$ be a tree-decomposition of G and let G' be the chordalization of G given by $\mathcal{T}$. Furthermore let k be the minimum number of maximal cliques of G' needed to separate all tight asteroidal triples in G' and t be the number of

tight asteroidal triples in G'. Then either $\mathcal{T}$ has between k and $6t$ joins or we can find a tree-decomposition $\mathcal{T}'$ of G in polynomial time with the same width as $\mathcal{T}$ and between k and $6t$ joins. Thus the result is not just a result on chordal graphs, but on general graphs.

Second, since we showed the existence of a $6t$-approximation algorithm constructively, we can use that algorithm to build a heuristic that finds tree-decomposition for general graphs and tries to minimize width and number of joins:

1. Let G be a graph. Use one of the many know algorithms that try to find tree-decomposition $\mathcal{T}$ of G with small width.

2. Calculate the chordalization G' of G given by $\mathcal{T}$. Use Algorithms 3.1 to try to minimize the number of joins without increasing treewidth.

These seem to be clear indications that the notion of tight asteroidal triples will be useful for many problems where the structure of tree-decompositions is analyzed, i.e. a difference between introduce and join bags is made.

4.2 FUTURE WORK

In Chapter 3 we presented three conjectures:

- The problem of finding a natural tree-decomposition with a minimum number of joins is NP-hard.

- If k is the minimum number of maximal cliques needed to separate all tight asteroidal triples of a chordal graph G, then there is a tree-decomposition of G with k joins. This would mean that our lower bound is tight.

- Algorithm 3.1 has a run that generates a natural tree-decomposition of a chordal graph with a minimum number of joins.

These are all interesting questions and can be a good starting point for further research.

It would also be interesting run experiments on Algorithms 3.1 and 3.3 to see how well they work in practice. For Algorithm 3.1 it would be specially interesting to see how much the heuristic improvements proposed in Section 3.1.1 help.

There are several other doors that tight asteroidal triples open. It may be interesting to develop an algorithm to find tree-decompositions of general graphs that takes into account

how many tight asteroidal triples it is generating in the underlying chordalization. This would increase the chances of Algorithms 3.1 to work well on the resulting tree-decomposition.

Furthermore, during the course of Chapter 3 we showed several properties that tight asteroidal triples must posses in chordal graphs. It is possible that some heuristics for tree-decompositions of small width can be improved by taking into account these properties of tight asteroidal triples. Theorem 3.44 seems to indicate that minimizing width means maximizing number of joins. It could be that e.g. guessing which are triples are going to be tight asteroidal triple and adapting the structure of the graph to their known features before using an already known heuristic for treewidth could improve its results. It may even be possible to find some tight asteroidal triples in a general graph. Maybe it is a good idea to try to keep them as tight asteroidal triples.

In Section 3.2.1 we gave an idea for an operation on which to base an algorithm that would manipulate tree-decompositions and try find a tree-decomposition of the graph with smaller join bags, at the necessary cost of increasing the size of some introduce bags. It may be very useful for practice to develop a heuristic based on this operation for the following problem:

MinRunningTime for chordal graphs

Input: A graph G, a tree-decomposition $\mathcal{T}$ of G, an integer s_j giving the maximum size of join bags and an integer s_i giving a maximum size for introduce bags, where $s_j \leq s_i$.

Problem: Let G' be the chordalization of G given by $\mathcal{T}$. Find a tree-decomposition of G' where the size of the join bags is smaller than s_j and the size of the introduce bags is smaller than s_i.

By choosing the appropriate variables s_i and s_j one could express the difference in running time of normal tree-decompositions algorithms on join and introduce bags.

Finally one could attempt to tackle a much more complicated problem:

MAXINTRODUCEANDJOIN FOR CHORDAL GRAPHS

Input: A graph G, And one introduce function and one join function $i, j : \mathbb{N} \mapsto \mathbb{R}$

Problem: Find the tree-decomposition $\mathcal{T}$ of G where the sum

$$\sum_{I \in \mathcal{I}(\mathcal{T})} i(|I|) + \sum_{J \in \mathcal{J}(\mathcal{T})} j(|J|)$$

is minimal.

This definition of optimality will be, if i and j are relatively tight, very close to optimizing the actual running time of a normal tree-decomposition algorithm. This problem seems to be much harder then the other problem we have looked at. This does not mean that an heuristic for this problem could not be useful and work well for certain instances or practical cases.

It seems that there are a lot of venues open for further research on the structure of tree-decompositions.

BIBLIOGRAPHY

[1] S. Arnborg and A. Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11 – 24, 1989.

[2] H. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 105–118. Springer Berlin / Heidelberg, 1988.

[3] H. Bodlaender. Discovering treewidth. In *SOFSEM 2005: Theory and Practice of Computer Science*, volume 3381 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin / Heidelberg, 2005.

[4] H. Bodlaender. Treewidth: Characterizations, applications, and computations. In *Graph-Theoretic Concepts in Computer Science*, volume 4271 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin / Heidelberg, 2006.

[5] H. Bodlaender and J. Engelfriet. Domino treewidth. In Ernst Mayr, Gunther Schmidt, and Gottfried Tinhofer, editors, *Graph-Theoretic Concepts in Computer Science*, volume 903 of *Lecture Notes in Computer Science*, pages 1–13. Springer Berlin / Heidelberg, 1995.

[6] H. Bodlaender and F. Fomin. Tree decompositions with small cost. In Martti Penttonen and Erik Schmidt, editors, *Algorithm Theory – SWAT 2002*, volume 2368 of *Lecture Notes in Computer Science*, pages 215–236. Springer Berlin/Heidelberg, 2002.

[7] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[8] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.

[9] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

[10] Hans Bodlaender. Treewidth: Algorithmic techniques and results. In Igor Prívara and Peter Ružicka, editors, *Mathematical Foundations of Computer Science 1997*, volume 1295

of *Lecture Notes in Computer Science*, pages 19–36. Springer Berlin / Heidelberg, 1997.

[11] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca. On treewidth approximations. *Discrete Applied Mathematics*, 136(2-3):183 – 196, 2004.

[12] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. SIAM monographs on discrete mathematics and applications. SIAM, 1999.

[13] D. G. Corneil, S. Olariu, and L. Stewart. Asteroidal triple-free graphs. *SIAM J. Discret. Math.*, 10:399–430, July 1997.

[14] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, 2001.

[15] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87:873–921, 1992.

[16] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141:109–131, 1995.

[17] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[18] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47 – 56, 1974.

[19] J. Gusted. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3):233 – 248, 1993.

[20] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Communications*, pages 85–103. Plenum Press, 1972.

[21] T. Kloks, D. Kratsch, and J. Spinrad. On treewidth and minimum fill-in of asteroidal triple-free graphs. *Theoretical Computer Science*, 175(2):309 – 335, 1997.

[22] C. G. Lekkerkerker and J. Ch. Boland. Representation of a finite graph by a set of intervals on the real line. volume 51, pages 45–64, 1962.

[23] A. Parra and P. Scheffler. How to use the minimal separators of a graph for its chordal triangulation. In *Automata, Languages and Programming*, volume 944 of *Lecture Notes in Computer Science*, pages 123–134. Springer Berlin / Heidelberg, 1995.

[24] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.

[25] D. J. Rose and R. E. Tarjan. Algorithmic aspects of vertex elimination. In *Proceedings of seventh annual ACM symposium on Theory of computing*, STOC '75, pages 245–254, New York, NY, USA, 1975. ACM.

[26] K. Suchan and I. Todinca. Pathwidth of circular-arc graphs. In *Graph-Theoretic Concepts in Computer Science*, volume 4769 of *Lecture Notes in Computer Science*, pages 258–269. Springer Berlin / Heidelberg, 2007.

[27] J. A. Telle. Tree-decompositions of small pathwidth. *Discrete Applied Mathematics*, 145(2):210 – 218, 2005.

[28] S. van Hoesel and B. Marchal. Finding good tree decompositions by local search. *Electronic Notes in Discrete Mathematics*, 32:43 – 50, 2009.