MSc thesis

Computer Science

Finding Optimal Tree Decompositions

Tuukka Korhonen

June 4, 2020

Faculty of Science

University of Helsinki

Supervisor(s)

Assoc. Prof. Matti Järvisalo, Dr. Jeremias Berg

Examiner(s)

Assoc. Prof. Matti Järvisalo, Dr. Jeremias Berg

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Computer Science

Tekijä — Författare — Author

Tuukka Korhonen

Työn nimi — Arbetets titel — Title

Finding Optimal Tree Decompositions

Ohjaajat — Handledare — Supervisors

Assoc. Prof. Matti Järvisalo, Dr. Jeremias Berg

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc thesis	June 4, 2020	94 pages, 1 appendix page

Tiivistelmä — Referat — Abstract

The task of organizing a given graph into a structure called a tree decomposition is relevant in multiple areas of computer science. In particular, many NP-hard problems can be solved in polynomial time if a suitable tree decomposition of a graph describing the problem instance is given as a part of the input. This motivates the task of finding as good tree decompositions as possible, or ideally, optimal tree decompositions.

This thesis is about finding optimal tree decompositions of graphs with respect to several notions of optimality. Each of the considered notions measures the quality of a tree decomposition in the context of an application. In particular, we consider a total of seven problems that are formulated as finding optimal tree decompositions: treewidth, minimum fill-in, generalized and fractional hypertreewidth, total table size, phylogenetic character compatibility, and treelength. For each of these problems we consider the BT algorithm of Bouchitté and Todinca as the method of finding optimal tree decompositions.

The BT algorithm is well-known on the theoretical side, but to our knowledge the first time it was implemented was only recently for the 2nd Parameterized Algorithms and Computational Experiments Challenge (PACE 2017). The author's implementation of the BT algorithm took the second place in the minimum fill-in track of PACE 2017. In this thesis we review and extend the BT algorithm and our implementation. In particular, we improve the efficiency of the algorithm in terms of both theory and practice. We also implement the algorithm for each of the seven problems considered, introducing a novel adaptation of the algorithm for the maximum compatibility problem of phylogenetic characters. Our implementation outperforms alternative state-of-the-art approaches in terms of numbers of test instances solved on well-known benchmarks on minimum fill-in, generalized hypertreewidth, fractional hypertreewidth, total table size, and the maximum compatibility problem of phylogenetic characters. Furthermore, to our understanding the implementation is the first exact approach for the treelength problem.

ACM Computing Classification System (CCS)
Theory of computation → Design and analysis of algorithms → Graph algorithms analysis

Avainsanat — Nyckelord — Keywords

tree decompositions, triangulations, Bouchitté–Todinca algorithm, potential maximal cliques

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Algorithms specialisation line

Contents

1 Introduction

Graphs and trees are ubiquitous objects in computer science. Graphs are used to represent any data with pairwise relations, such as flights between cities, connections in social networks, or dependencies between variables. On the other hand, trees are used to represent more structured data, such as elements in a web page or indexed databases. Graphs allow the representation of more general structures than trees. However, this comes with the cost that many algorithmic problems solved efficiently in trees are intractable in graphs in general. For example, the graph problems of maximum independent set and minimum dominating set are NP-hard in general but can be solved in linear time if the input graph is a tree [5]. The desire to attain the best of both worlds motivates the task of organizing arbitrary graphs into tree-like structures.

A tree decomposition of a graph is a tree whose nodes correspond to bags that contain vertices of the graph and whose structure maintains certain connectivity properties of the graph [15]. Tree decompositions have been rediscovered multiple times [14, 57, 91] and are also known as join trees [8], clique trees [46], and junction trees [62] in different contexts. It is trivial to obtain a tree decomposition of a graph in the form of a single bag containing all vertices of the graph. However, such a tree decomposition does not give any insight on the structure of the graph, and is thus not useful. To quantify the usefulness of a tree decomposition, multiple different cost functions on the quality of tree decompositions have been introduced, each with the associated goal to find tree decompositions with as small cost as possible, or in other words, optimal tree decompositions.

Perhaps the most well-known cost function associated with tree decompositions is the width of a tree decomposition, measuring the number of vertices in the largest bag of the decomposition [15]. The graph parameter of treewidth denotes the smallest width of a tree decomposition of a graph [15]. The graphs with small treewidth generalize trees, trees having treewidth one. Many classical NP-hard graph problems, such as maximum independent set, minimum dominating set, chromatic number, and Hamiltonicity can be solved with dynamic programming over a tree decomposition, with a running time exponential in the width of the decomposition but linear in the size of the graph [5]. The areas in which tree decompositions with small width have been applied in practice include probabilistic inference [31], propositional model counting [39], and register allocation in compilers [74].

The notion of treewidth has been extended in multiple directions. In the context of probabilistic inference, the total table size of a tree decomposition measures the running time of the junction tree algorithm using the decomposition more accurately than the treewidth [62, 67, 83]. Generalized and fractional hypertreewidth extend the notion of treewidth to hypergraphs, opening up more islands of tractability in constraint satisfaction problems [50, 52] and having applications in database systems [1, 66]. Moreover, problems

in phylogenetics on finding evolutionary trees have been formulated directly as finding optimal tree decompositions with respect to certain cost functions [21, 25].

This thesis is about finding optimal tree decompositions of graphs. We consider a total of seven graph problems, each defined as finding an optimal tree decomposition of a given graph with respect to a cost function. The problems we consider are treewidth [15], minimum fill-in [102], generalized hypertreewidth [50], fractional hypertreewidth [52], total table size [62, 83], phylogenetic character compatibility [21, 96], and treelength [34]. All of these problems are NP-hard [15, 21, 50, 52, 79, 80, 102]. To briefly describe the problems not mentioned in the previous paragraph, the goal in minimum fill-in is to minimize the number of so-called fill-edges of the decomposition, motivated by the task of finding an ordering for variable elimination in sparse matrices that minimizes the amount of extra memory required [92, 102]. Treelength is a more recent graph parameter, with applications in data structures for distance queries [34].

In particular, we focus on the BT algorithm of Bouchitté and Todinca [22, 23] as a method of finding optimal tree decompositions. The BT algorithm makes use of the theory of minimal triangulations, characterizing the minimal triangulations of a graph via objects called minimal separators and potential maximal cliques of the graph [23]. With respect to the cost functions considered in this thesis, the concept of minimal triangulations is roughly equivalent to the concept of tree decompositions for the purpose of finding optimal tree decompositions.

The BT algorithm was introduced in 2001 [22, 23]. Despite high interest from the theoretical side [16, 17, 42, 43, 44, 45, 86], the first implementations of the algorithm were introduced only recently in the 2nd Parameterized Algorithms and Computational Experiments Challenge (PACE 2017) [33]. PACE 2017 was an algorithm implementation competition whose topics included the exact computation of treewidth and minimum fill-in. Implementations based on the BT algorithm took the top three places in the minimum fill-in track and the second place in the treewidth track [33]. The author's implementation took the second place in the minimum fill-in track. After PACE 2017, we have developed our implementation by generalizing it to further problems [71, 72] and by considering alternative approaches for some parts of the algorithm [70]. In this thesis we introduce a further improved implementation of the BT algorithm for each of the seven problems.

In addition to generalizing and improving our implementation of the BT algorithm, we empirically compare the implementation to a total of 15 different alternative approaches for the problems. The alternative approaches are based on multiple different algorithmic ideas. The QuickBB algorithm for computing treewidth [48], the EDFS algorithm for computing total table size [77], and the FraSMT declarative approach for generalized and fractional hypertreewidth [38] make use of the elimination ordering characterization of triangulations [93]. The MCCP approach for minimum fill-in [10] and two approaches for phylogenetic character compatibility [53, 97] use problem-specific integer programming encodings. Other alternative approaches include implementations of fixed parameter algorithms [26], slice-wise polynomial-time algorithms [33, 40, 51], and alternative variants of the BT algorithm [33, 99].

In addition to the competition entries in PACE 2017 [33], the other experimental works on the BT algorithm that we are aware of are variants for computing treewidth by Tamaki [98, 99] and an adaptation for enumerating minimal triangulations by Ravid et al. [89]. On the theory side, the BT algorithm has been used to yield the best known time complexities for treewidth, minimum fill-in, fractional hypertreewidth, total table size, and treelength, yielding $O(1.7347^n)$-time algorithms for each of the problems [16, 43, 80, 86]. We remark that our implementation does not respect the $O(1.7347^n)$ time bound because we use a different, arguably more practical, subroutine for enumerating potential maximal cliques. On the other hand, our implementation realizes the theoretical result that treewidth, minimum fill-in, total table size, and treelength can be computed in polynomial time in graph classes with a polynomial number of minimal separators [22, 23].

1.1 Contributions

This thesis is mainly based on the author's articles [71, 72] and highly related to the author's article [70]. The original contributions of these articles presented in this thesis are an implementation of the BT algorithm for treewidth, minimum fill-in, generalized and fractional hypertreewidth, and total table size of Bayesian networks [71] and an adaptation of the BT algorithm for the maximum compatibility problem of multi-state phylogenetic characters and its implementation [72].

In addition to presenting the contributions of [71, 72], this thesis also includes further contributions. We introduce a new implementation of the BT algorithm, which in comparison to our earlier implementations includes new problem-specific variants of BT for perfect phylogeny, the maximum compatibility problem of binary phylogenetic characters, and treelength. We also improve the efficiency of the implementation in general, resulting in the new implementation being faster than the earlier implementations on all of the problems considered. Especially, we introduce a novel modification of the part of the BT algorithm that enumerates the potential maximal cliques of a graph. In addition to practical improvements, this modification reduces the upper bound of the number of potential maximal cliques of a graph from $\Delta^2 n + \Delta n + 1$ [22] to $\Delta^2 + \Delta n + 1$ and the time complexity of enumerating them from $O(\Delta^2 n^2 m)$ [22] to $O(\Delta^2 nm + \Delta n^2 m)$, where n is the number of vertices, m is the number of edges, and Δ is the number of minimal separators of the graph (Theorem 3).

In the experimental evaluation, our implementation of the BT algorithm solves more test instances within a 1-hour time limit than any alternative approach on multiple problems. On both generalized and fractional hypertreewidth, our implementation solves over 20% more instances on the Hyperbench dataset [40] than any other approach. On the maximum compatibility problem of multi-state phylogenetic characters, our implementation outperforms other approaches in scalability on generated data with respect to multiple parameters underlying the problem. On minimum fill-in, our implementation solves 8 instances more on a dataset of 330 instances in comparison to the algorithm that took

the first place in PACE 2017 [33] and significantly more instances than other approaches. On total table size, our implementation solves as many instances as the previous state-of-the-art [77] but does so several orders of magnitude faster. Furthermore, to the best of our knowledge our implementation is the first implementation for exact computing of treelength.

1.2 Organization of the Thesis

In Chapter 2 we present necessary background on tree decompositions and triangulations and the generic reduction from finding optimal tree decompositions to finding optimal minimal triangulations. In Chapter 3 we briefly review the background of each of the problems and the formulations of them in terms of finding optimal minimal triangulations. In Chapter 4 we review the BT algorithm for finding optimal minimal triangulations. We also introduce our novel modification of the part of the algorithm that enumerates potential maximal cliques and our novel adaptation of the algorithm for the maximum compatibility problem of multi-state phylogenetic characters. In Chapter 5 we introduce our implementation of the BT algorithm, presenting the preprocessing techniques that we use, further optimization for enumerating potential maximal cliques, and other implementation details. In Chapter 6 we empirically compare our implementation of the BT algorithm to alternative approaches for the problems. In Chapter 7 we further empirically analyze our implementation. Finally, we conclude the thesis in Chapter 8, also presenting ideas for future work.

2 Preliminaries

In this chapter we review background on tree decompositions and triangulations. First we recall necessary definitions and notations in graph theory. Then we define tree decompositions and state some of their key properties. Finally, we explain how the computation of optimal tree decompositions reduces to the computation of optimal minimal triangulations.

2.1 Graph Theory and Notations

We often consider collections of sets that have maximal and minimal elements. Unless otherwise specified, a set S in a collection $\mathcal{S}$ is maximal if there is no set $S' \in \mathcal{S}$ with $S \subset S'$. A set $S \in \mathcal{S}$ is minimal if there is no set $S' \in \mathcal{S}$ with $S' \subset S$.

We consider graphs that are finite, simple and undirected, unless otherwise stated. A graph G consists of a set of vertices $V(G)$ and a set of edges $E(G)$. The edges of a graph are unordered pairs of distinct vertices. When the graph G is clear from the context, we use $n = |V(G)|$ and $m = |E(G)|$ to denote the numbers of vertices and edges, respectively. A path between vertices v_1 and v_p is a sequence of distinct vertices $v_1, v_2, \ldots, v_p$, such that for each $1 \leq i < p$ there is an edge $\{v_i, v_{i+1}\} \in E(G)$. A cycle is a path $v_1, v_2, \ldots, v_p$ with $\{v_p, v_1\} \in E(G)$ and $p \geq 3$. A pair of vertices is connected if there is a path between them. A connected component $C \subseteq V(G)$ of a graph G is a maximal subset of vertices such that each pair of vertices $u, v \in C$ is connected. The set of connected components of a graph G is denoted by $\mathcal{C}(G)$. A graph is connected if it has exactly one connected component. A tree is a connected graph that has no cycles.

The neighborhood of a vertex v is $N(v) = \{u \mid \{v, u\} \in E(G)\}$ and the neighborhood of a vertex set X is $N(X) = \bigcup_{v \in X} N(v) \setminus X$. The closed neighborhood of a vertex v is $N[v] = N(v) \cup \{v\}$. The vertices $u \in N(v)$ are neighbors of v. For a vertex set X, the set of all possible edges within X is $X^2 = \{\{u, v\} \mid u, v \in X, u \neq v\}$. The induced subgraph $G[X]$ of a graph G, where $X \subseteq V(G)$, has $V(G[X]) = X$ and $E(G[X]) = E(G) \cap X^2$. We also use the notation $G \setminus X = G[V(G) \setminus X]$ to denote induced subgraphs. The vertex set X is a clique in a graph G if $E(G[X]) = X^2$.

2.2 Tree Decompositions

This thesis is about finding optimal tree decompositions of graphs. A property that makes trees useful in the design of algorithms is that they have no cycles, or equivalently, that there is an unique path between any two nodes of a tree. Tree decompositions generalize this concept to graphs. A *tree decomposition* of a graph G is a tree whose nodes are labeled

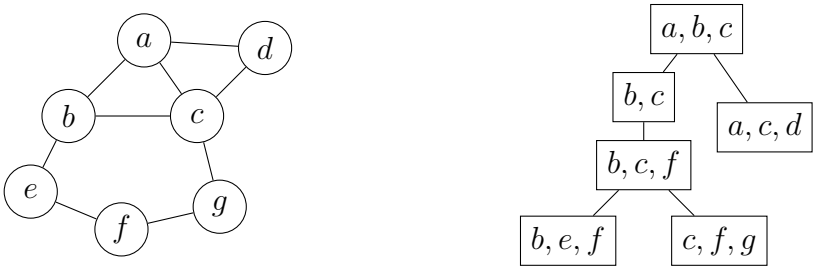

Figure 2.1: A graph (left) and one of its tree decompositions (right).

with *bags* that are subsets of $V(G)$. If u and v are nodes of a tree decomposition and the path between u and v goes through a node w, then all paths in G between any vertex of the bag of u and any vertex of the bag of v go through at least one vertex of the bag of w. More precisely, tree decompositions are defined as follows.

Definition 1 ([15])**.** *Let G be a graph, and consider a pair $(T, \mathcal{B})$ where T is a tree and $\mathcal{B}$ is a collection of bags that correspond to the nodes of T and contain vertices of G, i.e., $\mathcal{B} = \{B_i \mid i \in V(T)\}$ and $B_i \subseteq V(G)$ for all $i \in V(T)$. The pair $(T, \mathcal{B})$ is a tree decomposition of G if*

1. *$V(G) = \bigcup_{i \in V(T)} B_i$, i.e., each vertex is in a bag,*
2. *$E(G) \subseteq \bigcup_{i \in V(T)} B_i^2$, i.e., each edge is in a bag, and*
3. *for all $v \in V(G)$ the subgraph $T[\{i \mid v \in B_i\}]$ of T induced by bags that contain v is connected.*

Example 1. *Consider the graph G in Figure 2.1 (left) and one of its tree decompositions $(T, \mathcal{B})$ in Figure 2.1 (right). The bags of $(T, \mathcal{B})$ are $\mathcal{B} = \{\{a, b, c\}, \{b, c\}, \{b, c, f\}, \{b, e, f\}, \{c, f, g\}, \{a, c, d\}\}$, containing all vertices and edges of G and thus satisfying conditions 1 and 2 of Definition 1. We can also verify that $(T, \mathcal{B})$ satisfies condition 3 of Definition 1. For example, for the vertex $f \in V(G)$, the nodes corresponding to bags $\{b, e, f\}$, $\{b, c, f\}$ and $\{c, f, g\}$ form a connected subgraph of T. In the tree T, all paths between the node corresponding to bag $\{a, c, d\}$ and the node corresponding to bag $\{b, e, f\}$ go through the node corresponding to bag $\{b, c\}$. Correspondingly, all paths in the graph G from a or d to e or f go through b or c.*

In this thesis we consider multiple different cost functions that measure the quality of a tree decomposition. Perhaps the most well-known of such measures is the *width* of a tree decomposition. The width of a tree decomposition is $\mathtt{tw}(T, \mathcal{B}) = \max_{B_i \in \mathcal{B}} |B_i| - 1$, i.e., the size of the largest bag minus one. The *treewidth* of a graph G, $\mathtt{tw}(G)$, is the smallest possible width of a tree decomposition of G [15]. Note that a single bag that contains all vertices is always a tree decomposition, and therefore $\mathtt{tw}(G) \leq n - 1$.

Example 2. *Consider the graph G and one of its tree decompositions $(T, \mathcal{B})$ in Figure 2.1. The largest bag of $(T, \mathcal{B})$ has size 3. Therefore, $\mathtt{tw}(T, \mathcal{B}) = 3 - 1 = 2$, and thus $\mathtt{tw}(G) \leq 2$.*

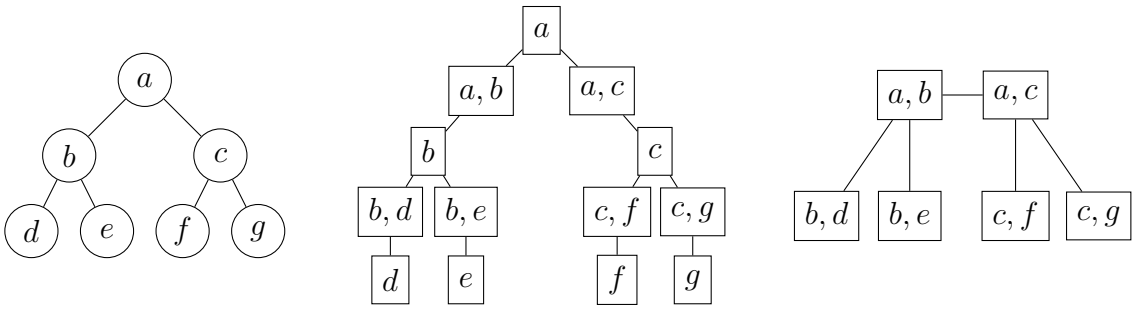

Figure 2.2: A tree (left) and two of its tree decompositions (middle and right).

The following two properties of tree decompositions can be used to deduce that the treewidth of the graph in Example 2 is exactly 2.

Proposition 1 ([15])**.** *If a graph G contains a clique $W \subseteq V(G)$, then all of its tree decompositions $(T, \mathcal{B})$ have a bag $B_i \in \mathcal{B}$ with $W \subseteq B_i$.*

Example 3. *Consider the graph G in Figure 2.1. As G contains a clique $\{a, b, c\}$, by Proposition 1 every tree decomposition of G has a bag that is a superset of $\{a, b, c\}$. Therefore, every tree decomposition of G has width at least 2, and thus $\texttt{tw}(G) = 2$.*

A forest is a graph whose all connected components are trees.

Proposition 2 ([15])**.** *A graph has treewidth at most 1 if and only if it is a forest.*

Example 4. *Consider the graph G in Figure 2.1. It has a cycle b, c, g, f, e, so it is not a forest, and thus it has treewidth at least 2. On the other hand, consider the tree T in Figure 2.2 (left). We can obtain a tree decomposition of T of width 1 by having a bag for each vertex and for each edge, as shown in Figure 2.2 (middle). This construction can be further simplified by removing the bags containing only single vertices, as shown in Figure 2.2 (right).*

The construction of Example 4 can be generalized to any tree. Note that it also applies to forests because tree decompositions of different connected components can be arbitrarily connected to each other without increasing treewidth.

2.3 Triangulations

Triangulations of graphs are a central graph-theoretic concept in the computation of tree decompositions. Triangulations are defined via chordality of graphs.

A graph G is *chordal* if every cycle in G with at least 4 vertices contains a *chord*, i.e., an edge between two non-adjacent vertices of the cycle [93]. Correspondingly, a non-chordal graph has at least one *chordless cycle*, i.e., a cycle with at least 4 vertices that does not have a chord.

Figure 2.3: A non-chordal graph (left) and one of its triangulations (right).

Example 5. *Consider the graph G in Figure 2.3 (left). The graph is not chordal because it contains the chordless cycle b, c, f, d. It also contains the chordless cycles a, b, c, f, e and a, b, d, f, e. The graph H in Figure 2.3 (right) also contains the cycle b, c, f, d. However, the cycle has a chord $\{b, f\}$ in H. All cycles of H with at least 4 vertices have chords, so H is a chordal graph.*

Definition 2 ([93])**.** *A graph H is a triangulation of a graph G if H is chordal, $V(G) = V(H)$, and $E(G) \subseteq E(H)$.*

In words, a triangulation H of a graph G is a chordal graph obtained by adding edges to G. The additional edges, i.e., edges in $E(H) \setminus E(G)$, are called *fill-edges*. All graphs have at least one triangulation because complete graphs are chordal, and any graph can be turned into a complete graph by adding edges.

Example 6. *Consider the graph G in Figure 2.3 (left) and the chordal graph H in Figure 2.3 (right). The graph H is a triangulation of G because it is chordal, it has the same vertex set as G, and the set of its edges includes all edges of G. The fill-edges of H with respect to G are $E(H) \setminus E(G) = \{\{a, f\}, \{b, f\}\}$.*

The concept of *elimination orderings* is closely related to chordal graphs and triangulations. An elimination ordering of a graph G is an ordering $\pi : [1 \dots n] \to V(G)$ of its vertices. A *perfect elimination ordering* of a graph G is an elimination ordering π such that for each vertex $v = \pi(i)$, the neighbors of v that appear after v in the ordering form a clique, i.e., for all $1 \leq i \leq n$ the set $N(\pi(i)) \cap \{\pi(j) \mid j > i\}$ is a clique [93].

Proposition 3 ([93])**.** *A graph is chordal if and only if it has a perfect elimination ordering.*

To give intuition on Proposition 3, let π be an elimination ordering and let us consider the process of removing vertices in the order $\pi(1), \pi(2), \dots, \pi(n)$. If the neighbors of $\pi(1)$ form a clique, then no chordless cycle can pass through $\pi(1)$, and thus the graph $G[\{\pi(1), \dots, \pi(n)\}]$ is chordal if and only if the graph $G[\{\pi(2), \dots, \pi(n)\}]$ is chordal. It also holds that any chordal graph has a vertex whose neighbors form a clique [93]. This process gives an algorithm to recognize if a graph is chordal. The algorithm repeatedly removes a vertex whose neighborhood is a clique, until either all vertices are removed (in which case the graph is chordal), or no vertices can be removed (in which case the graph

is not chordal) [93]. While this algorithm is not linear-time, there are multiple linear-time algorithms for finding perfect elimination orderings [93, 101].

Triangulations of graphs can be computed via elimination orderings. For a graph G and an elimination ordering π, the graph $G(\pi)$ is defined by the process of removing the vertices of G in the order given by π and in each step i adding edges to G so that $N(\pi(i)) \cap \{\pi(j) \mid j > i\}$ forms a clique. The graph $G(\pi)$ is a triangulation of G whose fill-edges are the edges added in this process [93]. Moreover, π is a perfect elimination ordering of $G(\pi)$.

Example 7. *Consider the graph G in Figure 2.3 (left) and one of its triangulations H in Figure 2.3 (right). An elimination ordering π with $\pi(1) = e$, $\pi(2) = a$, $\pi(3) = d$, $\pi(4) = b$, $\pi(5) = f$, $\pi(6) = c$ is a perfect elimination ordering of H. It is also an elimination ordering of G that defines the triangulation $G(\pi) = H$. Note that different elimination orderings can result in the same triangulation. In this case, for example, any order of the last 3 vertices in π would have resulted in the same triangulation H.*

A set of vertices is a maximal clique if it is a clique and no strict superset of it is a clique. The set of maximal cliques of a graph G is denoted by $\mathrm{MC}(G)$. The maximal cliques of a chordal graph have a special structure [46]. For any perfect elimination ordering π of a chordal graph H, each maximal clique $W \in \mathrm{MC}(H)$ is equal to $N[\pi(i)] \cap \{\pi(j) \mid j \geq i\}$ for some i [46]. In other words, each maximal clique appears in the elimination process as a closed neighborhood of a vertex that is being eliminated. It follows that a chordal graph has at most n maximal cliques.

Example 8. *Consider the chordal graph H in Figure 2.3 (right). The maximal cliques of H are $MC(H) = \{\{e, a, f\}, \{a, b, f\}, \{d, b, f\}, \{b, c, f\}\}$, listed in the order of finding them via the perfect elimination ordering e, a, d, b, f, c.*

2.4 From Tree Decompositions to Triangulations

We will now explain how finding a tree decomposition of a graph can be reduced to finding a triangulation of the graph. Triangulations and tree decompositions are connected by the fact that the maximal cliques of a chordal graph can be arranged into a tree so that they form a tree decomposition.

Proposition 4 ([46])**.** *Every chordal graph H has a tree decomposition $\mathtt{td}(H) = (T, \mathcal{B})$ with $\mathcal{B} = MC(H)$. Moreover, $\mathtt{td}(H)$ can be computed in linear time.*

The observation that makes Proposition 4 central in finding tree decompositions is that if H is a triangulation of a graph G, then any tree decomposition of H is a tree decomposition of G. In particular, $\mathtt{td}(H)$ is a tree decomposition of G.

Example 9. *Consider the graph G and one of its triangulations H in Figure 2.4 (left and middle). The maximal cliques of H are $MC(H) = \{\{a, b, c\}, \{b, c, e\}, \{b, d, e\}, \{c, e, f\}\}$.*

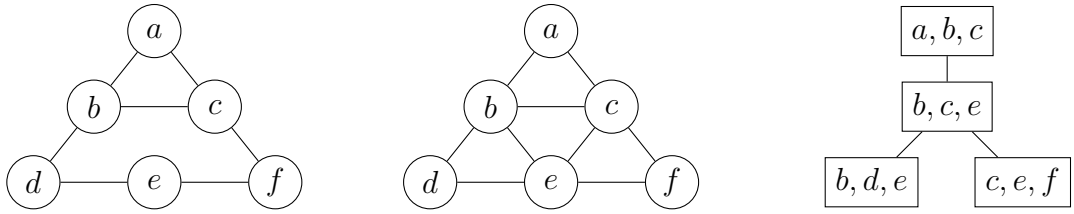

Figure 2.4: A graph (left), one of its triangulations (middle), and a tree decomposition that corresponds to the triangulation (right).

The tree decomposition $\texttt{td}(H)$ *shown in Figure 2.4 (right) has these cliques as its bags. As* H *is a triangulation of* G *and* $\texttt{td}(H)$ *is a tree decomposition of* H*,* $\texttt{td}(H)$ *is also a tree decomposition of* G*.*

Recall that by Proposition 1 each clique of a graph G must be contained in some bag of any tree decomposition of G. Therefore, the tree decomposition $\texttt{td}(H)$ given by Proposition 4 is optimal for H in the sense that each of its bags must appear as a bag or as a subset of a bag in every tree decomposition of H.

The algorithm for computing $\texttt{td}(H)$ in linear time first computes a perfect elimination ordering and then builds the tree decomposition starting from the leaves [46]. The intuition behind the algorithm is that eliminating a vertex corresponds to removing a leaf from the tree decomposition. The details of this algorithm are somewhat involved, so we omit them and refer the reader to [46] for details.

The notation $\texttt{td}(H)$ is a bit ambiguous since a chordal graph H can have multiple different (non-isomorphic) tree decompositions $\texttt{td}(H) = (T, \mathcal{B})$ with $\mathcal{B} = \mathrm{MC}(H)$ [46]. However, in the problems considered in this thesis the structure of T is not relevant apart from the fact that $(T, \mathcal{B})$ is a valid tree decomposition, so we will not assume any other properties of T. Note that, for example, the width of a tree decomposition depends only on $\mathcal{B}$.

By Proposition 4 a tree decomposition of a graph G can be computed by first computing a triangulation H of G and then computing a corresponding tree decomposition $\texttt{td}(H)$. The following proposition demonstrates that the tree decompositions arising from this process characterize all interesting tree decompositions.

Proposition 5 ([15])**.** *Let* $(T, \mathcal{B})$ *be a tree decomposition. The graph* $\texttt{ch}(T, \mathcal{B})$ *with the set of vertices* $V(\texttt{ch}(T, \mathcal{B})) = \bigcup_{i \in V(T)} B_i$ *and the set of edges* $E(\texttt{ch}(T, \mathcal{B})) = \bigcup_{i \in V(T)} B_i^2$ *is chordal and* $MC(\texttt{ch}(T, \mathcal{B})) \subseteq \mathcal{B}$*.*

An important observation following Proposition 5 is that if $(T, \mathcal{B})$ is a tree decomposition of a graph G, then $\texttt{ch}(T, \mathcal{B})$ is a triangulation of G. The notable difference in the transformations $\texttt{td}(H)$ and $\texttt{ch}(T, \mathcal{B})$ is that when transforming a tree decomposition into a chordal graph, some bags of the tree decomposition might not be maximal cliques of the chordal graph. This happens when $\mathcal{B}$ contains non-maximal bags which subsequently become non-maximal cliques of the chordal graph. However, one may argue that tree decompositions with non-maximal bags are not interesting. In particular, if $B_i \in \mathcal{B}$ is a

non-maximal bag, it has a neighbor $B_j \in \mathcal{B}$ with $B_i \subseteq B_j$ and the node i can be merged into the node j in the tree decomposition, yielding a tree decomposition with a set of bags $\mathcal{B} \setminus \{B_i\}$. Note that this does not change the width of the tree decomposition nor does it negatively affect any other cost function on tree decompositions that we consider in this thesis.

Putting together Propositions 4 and 5, we can conclude that triangulations of graphs can be used to compute exactly those tree decompositions that have only maximal bags. Furthermore, only these tree decompositions are interesting in our context. Next we detail an even smaller set of tree decompositions that is sufficient for finding optimal tree decompositions.

2.5 Cost Functions on Minimal Triangulations

A triangulation H of a graph G is a minimal triangulation if no triangulation H' of G has $E(H') \subset E(H)$ [93]. The set of minimal triangulations is often significantly smaller than the set of all triangulations. For example, a chordal graph has exactly one minimal triangulation but may have an exponential number of triangulations in general. In this section we detail how minimal triangulations are sufficient for finding optimal tree decompositions with respect to many different cost functions. The BT algorithm makes use of the combinatorial properties of minimal triangulations, characterizing exactly the minimal triangulations of a graph [23].

Example 10. *Consider the graph G in Figure 2.5 (left) and one of its triangulations H in Figure 2.5 (middle). The triangulation H is minimal because it has only one fill edge $\{a, e\}$ and G is not a chordal graph. Consider an other triangulation H' of G illustrated in Figure 2.5 (right). The triangulation H' is not minimal because $E(H) \subset E(H')$.*

Computing treewidth is equivalent to finding a triangulation where the largest clique is as small as possible [15]. It is straightforward to show that we can consider only minimal triangulations for this purpose. In particular, if H and H' are triangulations with $E(H) \subseteq E(H')$, then all cliques of H are also cliques of H', and therefore the largest clique of H is not larger than the largest clique of H'.

The argument for treewidth can be generalized to a large class of cost functions on triangulations. For a graph G, let $\mathrm{Tr}(G)$ denote the set of triangulations of G and $\mathrm{MT}(G)$ denote

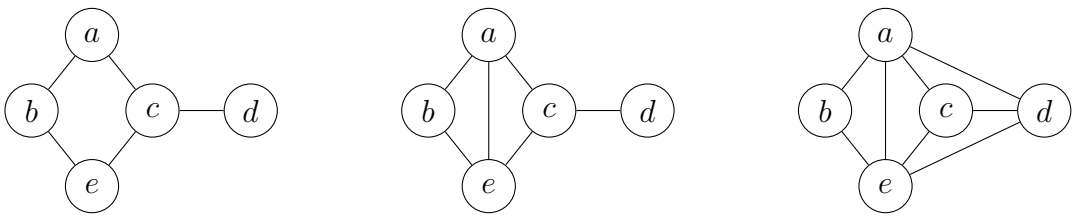

Figure 2.5: A graph (left), one of its minimal triangulations (middle), and one of its non-minimal triangulations (right).

the set of minimal triangulations of G. For any set X, let $\mathcal{P}(X)$ denote its power set, i.e., the set of all subsets of X. We detail three types of cost functions on triangulations with the property that there is a minimal triangulation that is optimal, i.e., takes the minimum value of the cost function over all triangulations.

Definition 3 (**Clique-max-type** [45])**.** *Let G be a graph and $f : \mathcal{P}(V(G)) \to \mathbb{R}_{\geq 0}$ a function that is monotone with respect to set inclusion, i.e., $f(W) \leq f(W')$ for $W \subseteq W'$. The function $C : Tr(G) \to \mathbb{R}_{\geq 0}$ with*

$$C(H) = \max_{W \in MC(H)} f(W)$$

is a clique-max-type cost function with a clique function f.

Computing the treewidth of a graph can be formulated as finding an optimal triangulation with respect to a clique-max-type cost function with a clique function $f(W) = |W|-1$ [45].

Definition 4 (**Fill-in-type** [45])**.** *Let G be a graph and $f : V(G)^2 \to \mathbb{R}_{\geq 0}$ a function. The function $C : Tr(G) \to \mathbb{R}_{\geq 0}$ with*

$$C(H) = \sum_{e \in E(H) \setminus E(G)} f(e)$$

is a fill-in-type cost function with a fill-edge function f.

To define the third type of cost functions, we make use of *fast functions*. A function $f : \mathcal{P}(X) \to \mathbb{R}_{\geq 0}$ is fast if for all $Y \subseteq X$ and $y \in Y$, $f(Y) \geq 2f(Y \setminus \{y\})$ [16]. A fast function $f(Y)$ grows fast when new items are added to the set Y. For example, a function $f(Y) = c^{|Y|}$ is fast for all $c \geq 2$.

Definition 5 (**Clique-sum-type** [16])**.** *Let G be a graph and $f : \mathcal{P}(V(G)) \to \mathbb{R}_{\geq 0}$ a fast function. The function $C : Tr(G) \to \mathbb{R}_{\geq 0}$ with*

$$C(H) = \sum_{W \in MC(H)} f(W)$$

is a clique-sum-type cost function with a clique function f.

All cost functions on triangulations that are clique-max-type, fill-in-type, or clique-sum-type admit optimal minimal triangulations.

Proposition 6 ([16, 45])**.** *Let G be a graph and $C : Tr(G) \to \mathbb{R}_{\geq 0}$ a clique-max-type, fill-in-type, or clique-sum-type cost function. There is a minimal triangulation $H \in MT(G)$ such that $C(H) \leq C(H')$ for all $H' \in Tr(G)$.*

The proof of Proposition 6 for clique-max-type follows our earlier sketch for treewidth. For fill-in-type it is straightforward from the definition that a fill-edge function outputs non-negative values. For clique-sum-type the proof is more complex. The intuition is that if we remove an edge e from triangulation H, each maximal clique containing e breaks down to at most two smaller maximal cliques [16]. We note that the constant 2 in the definition of a fast function is the smallest possible. In particular, for a cost function optimizing the sum of $1.99^{|W|}$ over maximal cliques $W \in \mathrm{MC}(H)$ all optimal triangulations may be non-minimal [16].

3 Optimal Tree Decompositions

In this thesis we apply the Bouchitté–Todinca algorithm for solving problems that are defined as finding optimal tree decompositions with respect to some notion of optimality. In this chapter we define each of these problems, namely treewidth, minimum fill-in, generalized and fractional hypertreewidth, total table size of Bayesian networks, perfect phylogeny, maximum compatibility of phylogenetic characters, and treelength. We formulate the problems in terms of finding optimal minimal triangulations of graphs, using the three generic cost function types defined in Section 2.5. We remark that by Proposition 6, formulating a problem as finding a triangulation minimizing a cost function of one of these types implies that the problem can be solved by considering only minimal triangulations.

3.1 Treewidth

The treewidth of a graph is the smallest width of a tree decomposition of the graph [15]. Computing treewidth is NP-hard [4]. The definition of treewidth is motivated by the fact that many graph problems can be solved with dynamic programming over a tree decomposition by computing for each bag a set of partial solutions, the number of which depends only on the number of vertices in the bag [5]. Such algorithms typically require (super-)exponential time in the sizes of the bags but polynomial time in the size of the input, therefore making the size of the largest bag the main contributor in the complexity of the algorithm [5, 27, 91].

Linear-time algorithms on graphs of bounded treewidth exist for many classical NP-hard problems, such as maximum independent set, minimum dominating set, chromatic number, and Hamiltonicity [5]. Their existence for a broad class of problems has been formalized by the Courcelle's Theorem, stating that any decision problem on graphs defined in monadic second-order logic can be solved in $O(f(\mathtt{tw}(G))n)$ time in a graph G, where $f(\mathtt{tw}(G))$ depends only on the treewidth of the input graph G [27]. In practice, algorithms using tree decompositions with small width are used in many areas of computer science, including constraint satisfaction [49, 73], probabilistic inference [31], propositional model counting [39] and register allocation in compilers [74].

We already formulated treewidth with optimal minimal triangulations in Section 2.5. Let us re-state the formulation here for completeness.

Proposition 7 ([15])**.** *The treewidth of a graph G is*

$$\min_{H \in Tr(G)} \max_{W \in MC(H)} |W| - 1.$$

Therefore, treewidth can be formulated as finding an optimal triangulation with respect to a clique-max-type cost function with a clique function $f(W) = |W| - 1$.

3.2 Minimum Fill-In

The minimum fill-in of a graph is the smallest number of fill-edges of a triangulation of the graph [102], relating more directly with triangulations rather than with tree decompositions. Computing minimum fill-in is NP-hard [102].

The concept of minimum fill-in is motivated by sparse matrix computations [87, 92, 102]. In this context, the edges of a graph represent non-zero elements in a system of equations, and a triangulation of the graph corresponds to an order to eliminate variables from the system. The number of fill-edges measures the number of additional non-zero entries required in the variable elimination process [92]. Minimum fill-in has received significant attention in algorithmics, motivated also by various other potential applications [17, 42].

Proposition 8 ([102])**.** *The minimum fill-in of a graph G is*

$$\min_{H \in Tr(G)} |E(H) \setminus E(G)|.$$

Therefore, minimum fill-in can be formulated as finding an optimal triangulation with respect to a fill-in-type cost function with a fill-edge function $f(e) = 1$.

3.3 Generalized and Fractional Hypertreewidth

Generalized hypertreewidth and fractional hypertreewidth extend the notion of treewidth to hypergraphs, enabling efficient solving of larger classes of constraint satisfaction problems [50]. A constraint satisfaction problem (CSP) consists of a set of variables, a domain of values of the variables, and a set of constraints, each over a subset of the variables. The goal is find an assignment of the variables that satisfies all of the constraints [49]. Many problems can be expressed as CSP [40]. For example, the Boolean satisfiability problem is a special case of CSP [52].

A hypergraph $\mathcal{G}$ consists of a set of vertices $V(\mathcal{G})$ and a set of hyperedges $E(\mathcal{G})$ that are arbitrary subsets of the vertices [49]. The structure of a CSP can be expressed as a hypergraph, with variables corresponding to the vertices and constraints corresponding to the hyperedges [49]. Given a generalized hypertree decomposition with bounded width or a fractional hypertree decomposition with bounded width, any CSP corresponding to the hypergraph can be solved in polynomial time [50]. Applications of generalized and fractional hypertreewidth also arise from databases [40], where multiple systems using generalized/fractional hypertree decompositions have been implemented [1, 66]. The width notions of hypergraphs have further applications in combinatorial optimization [40].

The primal graph $P(\mathcal{G})$ of a hypergraph $\mathcal{G}$ is the graph with the same vertex set as the hypergraph, i.e., $V(P(\mathcal{G})) = V(\mathcal{G})$ and with edges corresponding to the pairs of vertices included in a common hyperedge, i.e., $E(P(\mathcal{G})) = \bigcup_{e \in E(\mathcal{G})} e^2$.

Definition 6. *A tree decomposition of a hypergraph $\mathcal{G}$ is a tree decomposition of its primal graph $P(\mathcal{G})$.*

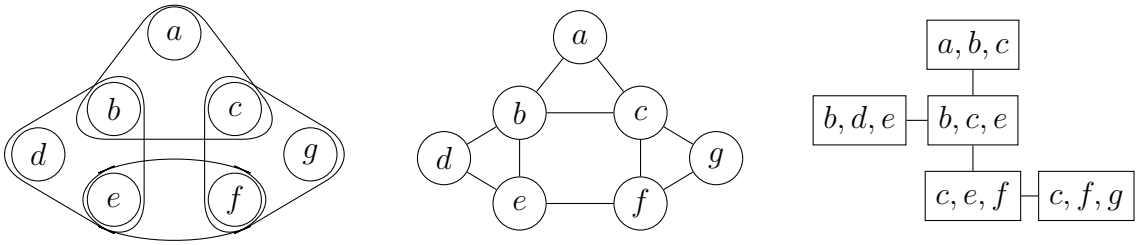

Figure 3.1: A hypergraph (left), its primal graph (middle), and one of its tree decompositions (right).

Definition 6 is equivalent to a natural generalization of tree decomposition of graphs (Definition 1) to hypergraphs, with condition 2 changed to $E(\mathcal{G}) \subseteq \cup_{i \in V(T)} \mathcal{P}(B_i)$. The equivalence follows from the fact that each hyperedge of $\mathcal{G}$ corresponds to a clique in $P(\mathcal{G})$, and by Proposition 1 each clique of a graph must be a subset of some bag in any tree decomposition of the graph.

Example 11. *Consider the hypergraph $\mathcal{G}$ in Figure 3.1 (left). Its vertex set is $V(\mathcal{G}) = \{a, b, c, d, e, f, g\}$ and its hyperedge set is $E(\mathcal{G}) = \{\{a, b, c\}, \{b, d, e\}, \{e, f\}, \{c, g, f\}\}$. Its primal graph $P(\mathcal{G})$ is shown in Figure 3.1 (middle). The primal graph of $\mathcal{G}$ has edges between all pairs of vertices that share a hyperedge in $\mathcal{G}$. A tree decomposition of $P(\mathcal{G})$ is shown in Figure 3.1 (right). It is also a tree decomposition of $\mathcal{G}$. For each hyperedge of $\mathcal{G}$, there is a bag in the tree decomposition that contains the hyperedge.*

3.3.1 Generalized Hypertreewidth

Generalized hypertreewidth is a generalization of the notion of treewidth to hypergraphs.

Definition 7 ([50])**.** *A generalized hypertree decomposition of a hypergraph $\mathcal{G}$ is a triple $(T, \mathcal{B}, \lambda)$, where $(T, \mathcal{B})$ is a tree decomposition of $\mathcal{G}$ and $\lambda = \{\lambda_i \mid i \in V(T)\}$ is a set of* edge covers *of bags, with $\lambda_i \subseteq E(\mathcal{G})$ satisfying $B_i \subseteq \cup_{e \in \lambda_i} e$. The width of a generalized hypertree decomposition is* $\max_{i \in V(T)} |\lambda_i|$.

In other words, a generalized hypertree decomposition is a tree decomposition that associates an edge cover to each bag, covering all vertices of the bag. The width of a generalized hypertree decomposition is the size of its largest edge cover. The generalized hypertreewidth of a hypergraph is the smallest width of its generalized hypertree decomposition [50].

Example 12. *Consider the hypergraph $\mathcal{G}$ in Figure 3.1 (left) and one of its tree decompositions $(T, \mathcal{B})$ in Figure 3.1 (right). A generalized hypertree decomposition of $\mathcal{G}$ can be obtained by associating each bag $B_i \in \mathcal{B}$ with an edge cover. Each of the bags $\{a, b, c\}$, $\{b, d, e\}$ and $\{c, f, g\}$ can be covered with a single edge, the bag $\{b, c, e\}$ can be covered with $\{\{a, b, c\}, \{b, d, e\}\}$, and the bag $\{c, e, f\}$ can be covered with $\{\{c, f, g\}, \{e, f\}\}$, resulting in a generalized hypertree decomposition with width 2. Therefore, the generalized hypertreewidth of $\mathcal{G}$ is at most 2.*

Let $\mathcal{G}$ be a hypergraph and $\mathrm{COV}(W)$ denote the size of smallest edge cover of a set $W \subseteq V(\mathcal{G})$, i.e., the size of smallest set $\lambda \subseteq E(\mathcal{G})$ with $W \subseteq \cup_{e \in \lambda} e$.

Proposition 9 ([86]). *The generalized hypertreewidth of a hypergraph $\mathcal{G}$ is*

$$\min_{H \in Tr(P(\mathcal{G}))} \max_{W \in MC(H)} COV(W).$$

The function $\mathrm{COV}(W)$ is monotone with respect to set inclusion because an edge cover of a set is also an edge cover of any of its subsets. Therefore, generalized hypertreewidth can be formulated as finding an optimal triangulation with respect to a clique-max-type cost function with a clique function $f(W) = \mathrm{COV}(W)$.

3.3.2 Fractional Hypertreewidth

Fractional hypertreewidth is a further generalization of the notion of generalized hypertreewidth.

Definition 8 ([52]). *A fractional hypertree decomposition of a hypergraph $\mathcal{G}$ is a triple $(T, \mathcal{B}, \lambda)$, where $(T, \mathcal{B})$ is a tree decomposition of $\mathcal{G}$ and $\lambda = \{\lambda_i \mid i \in V(T)\}$ is a set of* fractional edge covers *of the bags, with $\lambda_i : E(\mathcal{G}) \to [0, 1]$ satisfying for each vertex $v \in B_i$ of each bag $B_i \in \mathcal{B}$ that $\sum_{v \in e \in E(\mathcal{G})} \lambda_i(e) \geq 1$. The width of a fractional hypertree decomposition is $\max_{i \in V(T)} \sum_{e \in E(\mathcal{G})} \lambda_i(e)$.*

In other words, a fractional hypertree decomposition is a generalized hypertree decomposition, where the definition of edge cover is relaxed in the sense that edges can participate in the edge cover "fractionally". The fractional hypertreewidth of a hypergraph is the smallest width of its fractional hypertree decomposition [52]. Note that any generalized hypertree decomposition is also a fractional hypertree decomposition with the same width, and therefore fractional hypertreewidth is bounded from above by generalized hypertreewidth.

Example 13. *Consider the hypergraph $\mathcal{G}$ in Figure 3.2 (left) and one of its tree decompositions $(T, \mathcal{B})$ in Figure 3.2 (right). A fractional hypertree decomposition of $\mathcal{G}$ can be obtained by associating each bag $B_i \in \mathcal{B}$ with a fractional edge cover. Each of the bags $\{a, b, c\}$, $\{b, d, e\}$, and $\{c, e, f\}$ can be covered with a (fractional) edge cover of size one, but more interestingly the bag $\{b, c, e\}$ cannot be covered with a fractional edge cover of size one but can be covered with a fractional edge cover of size* 1.5 *by associating each hyperedge of $\mathcal{G}$ with the value* 0.5*. Therefore, the fractional hypertreewidth of $\mathcal{G}$ is at most* 1.5*. The fractional hypertreewidth of $\mathcal{G}$ is exactly* 1.5 *because $\{b, c, e\}$ is a bag or a subset of a bag in every tree decomposition of $\mathcal{G}$ but cannot be covered with a fractional edge cover of smaller size than* 1.5*.*

Let $\mathcal{G}$ be a hypergraph and $\mathrm{FCOV}(W)$ denote the size of smallest fractional edge cover of a set $W \subseteq V(\mathcal{G})$, i.e., the smallest sum of values of a function $\lambda : E(\mathcal{G}) \to [0, 1]$ with $\sum_{v \in e \in E(\mathcal{G})} \lambda(e) \geq 1$ for all $v \in W$.

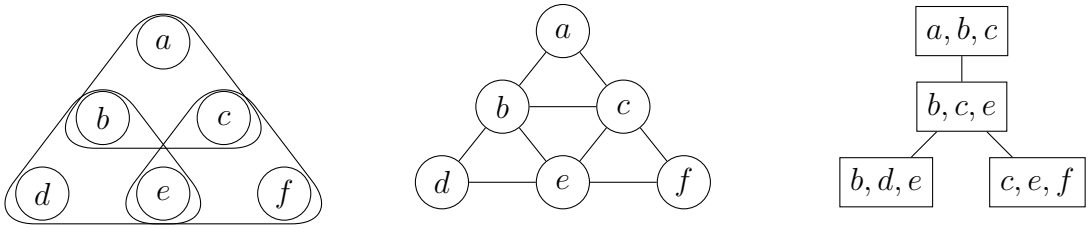

Figure 3.2: A hypergraph (left), its primal graph (middle), and one of its tree decompositions (right).

Proposition 10 ([86])**.** *The fractional hypertreewidth of a hypergraph $\mathcal{G}$ is*

$$\min_{H \in Tr(G)} \max_{W \in MC(H)} FCOV(W).$$

The function FCOV(W) is monotone with respect to set inclusion because a fractional edge cover of a set is also a fractional edge cover of any of its subsets. Therefore, fractional hypertreewidth can be formulated as finding an optimal triangulation with respect to a clique-max-type cost function with a clique function $f(W) = \text{FCOV}(W)$.

3.3.3 On the Notion of Hypertreewidth

In addition to treewidth, generalized hypertreewidth and fractional hypertreewidth, *hypertreewidth* is another well-known width notion of hypergraphs [49]. We review the definitions of hypertree decomposition and hypertreewidth and discuss why minimal triangulations cannot be directly applied to find optimal hypertree decompositions.

A hypertree decomposition of a hypergraph $\mathcal{G}$ is a generalized hypertree decomposition $(T, \mathcal{B}, \lambda)$ of $\mathcal{G}$ with one node $r \in V(T)$ chosen as a root and with one additional condition. Let $\mathcal{B}(T_i)$ denote the vertices of $\mathcal{G}$ that are in bags in the subtree of the node i with respect to the root, i.e., nodes j such that all paths from j to r go through i. The additional condition requires that the edge cover λ_i of the bag B_i should not cover any vertices that are in $\mathcal{B}(T_i)$ but not in B_i, i.e., $\cup_{e \in \lambda_i} e \cap \mathcal{B}(T_i) = B_i$ [49]. The width of a hypertree decomposition is the size of the largest edge cover and the hypertreewidth of a hypergraph is the smallest width of its hypertree decomposition [49].

All hypertree decompositions are generalized hypertree decompositions. Furthermore, it holds that $\mathtt{ghw}(\mathcal{G}) \leq \mathtt{hw}(\mathcal{G}) \leq 3\mathtt{ghw}(\mathcal{G}) + 1$ [3], where $\mathtt{ghw}(\mathcal{G})$ denotes the generalized hypertreewidth of a hypergraph $\mathcal{G}$ and $\mathtt{hw}(\mathcal{G})$ the hypertreewidth of a hypergraph $\mathcal{G}$. The motivation for the additional condition in the definition of hypertreewidth is that there is an $O(n^{2k+2})$-time algorithm for deciding if the hypertreewidth of a hypergraph is at most k [51]. Deciding if generalized hypertreewidth is at most k is NP-complete even for $k = 2$ [41]. However, hypertreewidth does not behave particularly well in the context of minimal triangulations.

Proposition 11. *There is a hypergraph $\mathcal{G}$ such that no tree decomposition $(T, \mathcal{B})$ that corresponds to a minimal triangulation of its primal graph can be extended to a hypertree decomposition $(T, \mathcal{B}, \lambda)$ of $\mathcal{G}$.*

Proof. Consider the hypergraph $\mathcal{G}$ in Figure 3.2 (left). Its primal graph $P(\mathcal{G})$ shown in Figure 3.2 (middle) is chordal, thus having $P(\mathcal{G})$ itself as its only minimal triangulation. The primal graph $P(\mathcal{G})$ defines a tree decomposition $\texttt{td}(P(\mathcal{G}))$ shown in Figure 3.2 (right). With any choice of the root, the bag $\{b, c, e\}$ of $\texttt{td}(P(\mathcal{G}))$ has at least two children. By symmetry, assume that the children are $\{b, d, e\}$ and $\{c, e, f\}$. In order to cover the vertex e of $\{b, c, e\}$, one of the hyperedges $\{b, d, e\}$ and $\{c, e, f\}$ must be included in the cover. However, including either would violate the additional condition of hypertree decompositions. □

By Proposition 11, tree decompositions corresponding to non-minimal triangulations must be considered in order to compute the hypertreewidth, which limits the applicability of the BT algorithm. Therefore we will not consider computing hypertreewidth in this thesis. However, in Chapter 6 we will compare our implementations of the BT algorithm for computing generalized and fractional hypertreewidth to an implementation that computes hypertreewidth. To the best of our knowledge, the notions of hypertree decomposition and generalized hypertree decomposition are interchangeable in all known applications that require such decompositions as a part of the input.

3.4 Total Table Size

Many algorithms that use tree decompositions consider for each bag of the tree decomposition a number of configurations that is exponential in the size of the bag [5, 62, 63]. If we know how the number of configurations depends on the tree decomposition, we may find a tree decomposition that explicitly minimizes this quantity instead of using the treewidth as a proxy for the quality of the tree decomposition. We consider this idea specifically in the context of Bayesian networks, where the total table size is a well-known measure of the quality of a tree decomposition [67, 77, 83].

A Bayesian network is a directed acyclic graph D, in which each vertex $v \in V(D)$ corresponds to a random variable X_v with a discrete state space and an associated conditional probability table $P(X_v \mid X_{p_1}, \ldots, X_{p_k})$ over the set of its parents $\{p_i \mid (p_i, v) \in E(D)\}$ [63]. In Bayesian networks, tree decompositions are used for probabilistic inference via the junction tree algorithm [62, 63]. A junction tree of a Bayesian network is essentially a tree decomposition of the *moral graph* of the Bayesian network [63].

Definition 9 ([63])**.** *The moral graph* $\texttt{mor}(D)$ *of a Bayesian network* D *is an undirected graph with* $V(\texttt{mor}(D)) = V(D)$ *and* $E(\texttt{mor}(D)) = \{\{v, u\} \mid (v, u) \in E(D)\} \cup \{\{u, w\} \mid (u, v), (w, v) \in E(D)\}$.

In words, the moral graph has the same vertex set as the Bayesian network, and its edge set includes the edges of the Bayesian network and additional edges between all pairs of parents of each vertex [63].

Example 14. *An example of a Bayesian network is shown in Figure 3.3 (left). This Bayesian network is from [76], where it was used to illustrate how the probability of a*

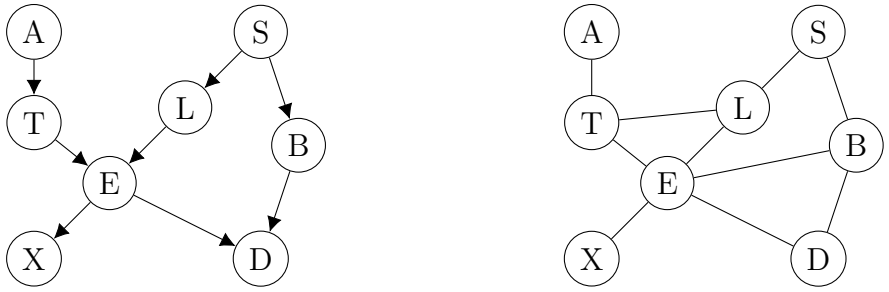

Figure 3.3: A Bayesian network from [76] (left) and its moral graph (right).

patient having tuberculosis (T), lung cancer (L), or bronchitis (B) could be inferred for example from the probabilities of the patient having visited Asia (A) and having a positive X-ray result (X). The arrows denote the directions of the edges of the Bayesian network. The moral graph of the Bayesian network is shown in Figure 3.3 (right). The moral graph is undirected and has the same vertex set as the Bayesian network. The edge set of the moral graph consist of the undirected versions of the edges of the Bayesian network and the additional edges $\{T, L\}$ and $\{E, B\}$ between the parents of E and D, respectively.

The complexity of the junction tree algorithm depends on the sizes of probability tables computed for each bag of the tree decomposition [62, 83]. For a tree decomposition $(T, \mathcal{B})$, this corresponds to total table size $\sum_{B_i \in \mathcal{B}} \prod_{v \in B_i} s(v)$, where $s(v) \geq 2$ denotes the number of states of the random variable X_v [67, 77, 83]. It is NP-hard to find a tree decomposition of a moral graph minimizing total table size [79].

Proposition 12 ([67, 77, 83])**.** *Let D be a Bayesian network and $s(v) \geq 2$ denote the number of states of the random variable associated with vertex $v \in V(D)$. The total table size of D is*

$$\min_{H \in Tr(\texttt{mor}(D))} \sum_{W \in MC(H)} \prod_{v \in W} s(v).$$

The function $\prod_{v \in W} s(v)$ is fast because $s(v) \geq 2$. Therefore total table size can be formulated as finding an optimal triangulation with respect to a clique-sum-type cost function with a clique function $f(W) = \prod_{v \in W} s(v)$ [16].

Remark 1. *In general there may be vertices $v \in V(D)$ with $s(v) = 1$. However, such vertices can be removed from the moral graph and later added to any tree decomposition without affecting its total table size.*

3.5 Phylogenetic Character Compatibility

In phylogenetics, a central problem is to find a phylogenetic tree that describes the evolution of a set of taxa[1] (species) given data about the taxa. Given a set of taxa and a set

[1]Singular: taxon.

of characters (attributes) which map the taxa to character states, the *perfect phylogeny problem* is to find a phylogenetic tree describing the taxa so that each character state evolves only once (or to report that no such tree exists) [96]. The *maximum compatibility problem* is to find the largest subset of the characters that admits a perfect phylogeny. These problems can be formulated via triangulations of graphs [21, 25].

We first define the perfect phylogeny problem and the maximum compatibility problem directly via phylogenetic trees and then show how they can be formulated via triangulations of partition intersection graphs of phylogenetic characters. The perfect phylogeny problem and the maximum compatibility problem restricted to binary characters can be formulated with cost functions of fill-in-type [55]. However, as we discussed in [72], the BT algorithm is not directly applicable for solving the maximum compatibility problem with characters of higher arity than two (multi-state characters). For solving the maximum compatibility problem in the general case of multi-state characters, we will introduce in Section 4.4 a hybrid algorithm that makes use of maximum satisfiability in conjunction with the BT algorithm.

3.5.1 Phylogenetic Trees

A character $\mathcal{X}$ on a set of taxa X is a function $\mathcal{X} : X'_{\mathcal{X}} \to C_{\mathcal{X}}$, where $X'_{\mathcal{X}}$ is a subset of X and $C_{\mathcal{X}}$ is the set of states of $\mathcal{X}$. A character $\mathcal{X}$ is an r-ary character if $|C_{\mathcal{X}}| = r$. In particular, $|C_{\mathcal{X}}| = 2$ for binary characters and $|C_{\mathcal{X}}| \geq 3$ for multi-state characters. A character is full if it maps all taxa to states, i.e., $X'_{\mathcal{X}} = X$ and partial otherwise. Note that an r-ary character $\mathcal{X}$ partitions $X'_{\mathcal{X}}$ into r parts. We denote this partition by $\pi(\mathcal{X}) = \{\mathcal{X}^{-1}\{\alpha\} \mid \alpha \in C_{\mathcal{X}}\}$.

A phylogenetic tree on a set of taxa X is a pair (T, ϕ), where T is a tree and $\phi : X \to V(T)$ maps the taxa to the nodes of the tree. We require that for each node $v \in V(T)$ with degree at most two there is a taxon that is mapped to v. For a set of nodes $S \subseteq V(T)$, let T_S be the minimal connected induced subgraph of T that contains all nodes in S. A phylogenetic tree displays a character $\mathcal{X}$ if no two of the trees in $\{T_{\phi(A)} \mid A \in \pi(\mathcal{X})\}$ intersect each other. In other words, the tree displays a character if the nodes of the tree can be partitioned into connected subtrees, each corresponding to exactly one state of the character.

Definition 10 ([96])**.** *A set of characters $\mathfrak{C}$ on taxa X is compatible if there is a phylogenetic tree on X that displays all characters in $\mathfrak{C}$.*

The perfect phylogeny problem is to decide if a set of characters is compatible and the maximum compatibility problem is to find a maximum size compatible subset of a given set of characters [96].

Example 15. *Consider the characters $\mathcal{X}_1, \ldots, \mathcal{X}_4$ on taxa $X_1, \ldots, X_5$ in Figure 3.4 (left). The characters are (at most) quaternary, with state set $\{A, C, G, T\}$. The character $\mathcal{X}_3$ is a partial character and the other characters are full. The tree in Figure 3.4 (right) is*

	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$	$\mathcal{X}_4$
X_1	A	C	A	A
X_2	C	T		G
X_3	G	C	G	G
X_4	T	A	G	A
X_5	A	T	T	C

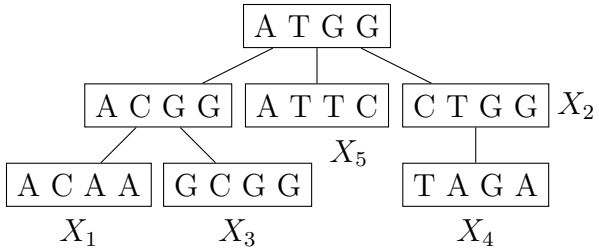

Figure 3.4: A table describing characters $\mathcal{X}_1, \ldots, \mathcal{X}_4$ on taxa $X_1, \ldots, X_5$ (left) and a phylogenetic tree on $X_1, \ldots, X_5$ displaying characters $\mathcal{X}_1, \mathcal{X}_2$ and $\mathcal{X}_3$ (right).

a phylogenetic tree on the taxa $X_1, \ldots, X_5$. It has five nodes corresponding to the taxa and two additional internal nodes. The internal nodes are labeled with the corresponding character-states. The tree displays the characters $\mathcal{X}_1, \mathcal{X}_2$ and $\mathcal{X}_3$ but not the character $\mathcal{X}_4$ because the subtrees $T_{\phi(\{X_1,X_4\})}$ and $T_{\phi(\{X_2,X_3\})}$, corresponding to the states A and G of $\mathcal{X}_4$, intersect. Therefore the set of characters $\{\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3\}$ is compatible.

3.5.2 Formulation by Triangulations

The perfect phylogeny problem and the maximum compatibility problem can be formulated in terms of triangulations of colored graphs. The graph whose triangulations we are interested in is the partition intersection graph (PI-graph) of a set of characters.

Definition 11 ([21, 25])**.** *Let $\mathfrak{C}$ be a set of characters. The partition intersection graph (PI-graph) int($\mathfrak{C}$) of $\mathfrak{C}$ has*

$$V(\mathit{int}(\mathfrak{C})) = \bigcup_{\mathcal{X} \in \mathfrak{C}} \{(\mathcal{X}, A) \mid A \in \pi(\mathcal{X})\} \quad \text{and}$$

$$E(\mathit{int}(\mathfrak{C})) = \{\{(\mathcal{X}, A), (\mathcal{X}', B)\} \mid A \cap B \neq \emptyset\}.$$

In words, the PI-graph of $\mathfrak{C}$ contains a vertex for each pair $(\mathcal{X}, A)$, where $\mathcal{X}$ is a character and $A \subseteq X$ is a part of the partition $\pi(\mathcal{X})$ induced by $\mathcal{X}$. The PI-graph has edges between the pairs of vertices whose corresponding parts of taxa have a non-empty intersection, hence the name "partition intersection graph". Note that there are no edges between two vertices corresponding to a same character because parts of a partition are disjoint. Note also that if all characters are r-ary, then the PI-graph has $|\mathfrak{C}|r$ vertices. The PI-graph can be thought of as being colored with $|\mathfrak{C}|$ colors, each vertex having a color corresponding to the character of the vertex. Initially, this is a valid coloring in the sense that there are no edges between vertices of the same color. The goal in finding triangulations of PI-graphs is to preserve as many colors as possible.

Proposition 13 ([21])**.** *Let $\mathfrak{C}$ be a set of characters. A subset $\mathfrak{C}' \subseteq \mathfrak{C}$ is compatible if and only if there exists a triangulation H of int($\mathfrak{C}$) that has no fill-edge of form $\{(\mathcal{X}, A), (\mathcal{X}, B)\} \in E(H) \setminus E(\mathit{int}(\mathfrak{C}))$ for any $\mathcal{X} \in \mathfrak{C}'$.*

	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$
X_1	1	0	1
X_2	1	1	
X_3	0	0	
X_4	0	1	0

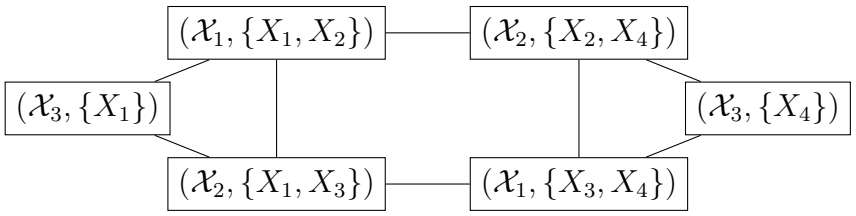

Figure 3.5: A table describing binary characters $\mathcal{X}_1, \ldots, \mathcal{X}_3$ on taxa $X_1, \ldots, X_4$ (left) and the PI-graph of them (right).

The intuition on how triangulations of PI-graphs relate to phylogenetic trees is that each maximal clique of a triangulation corresponds to a node of the phylogenetic tree.

Example 16. *Consider the characters $\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3$ on taxa $X_1, \ldots, X_4$ in Figure 3.5 (left). The PI-graph int($\{\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3\}$) is shown in Figure 3.5 (right). It has a vertex corresponding to each character-state pair of $\{\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3\}$. The PI-graph has exactly two minimal triangulations, one adding the fill-edge $\{(\mathcal{X}_1, \{X_1, X_2\}), (\mathcal{X}_1, \{X_3, X_4\})\}$ and another adding the fill-edge $\{(\mathcal{X}_2, \{X_1, X_3\}), (\mathcal{X}_2, \{X_2, X_4\})\}$. Therefore the characters do not admit a perfect phylogeny. The subsets $\{\mathcal{X}_2, \mathcal{X}_3\}$ and $\{\mathcal{X}_1, \mathcal{X}_3\}$ are maximum size subsets that are compatible.*

Proposition 13 allows to formulate the perfect phylogeny problem and the maximum compatibility problem of binary characters as fill-in-type. Let the fill-edge function $f(e)$ on potential fill-edges e of a PI-graph be

$$f(\{(\mathcal{X}_1, A), (\mathcal{X}_2, B)\}) = \begin{cases} 1 & \text{if } \mathcal{X}_1 = \mathcal{X}_2 \\ 0 & \text{otherwise} \end{cases} .$$

The function $f(e)$ takes value 1 if the edge e is between two vertices corresponding to the same character $\mathcal{X}$ and value 0 otherwise. By Proposition 13, the fill-edges e for which $f(e) = 1$ are exactly the fill-edges whose addition causes some characters to not be compatible.

Corollary 1 ([55])**.** *A set of characters $\mathfrak{C}$ admits a perfect phylogeny if and only if there exists a triangulation $H \in Tr(int(\mathfrak{C}))$ with*

$$\sum_{e \in E(H) \setminus E(int(\mathfrak{C}))} f(e) = 0.$$

By Corollary 1 perfect phylogeny can be formulated as finding an optimal triangulation with respect to a fill-in-type cost function with a fill-edge function $f(e)$. We note that perfect phylogeny can also be expressed as finding an optimal triangulation with respect to a clique-max-type cost function with a clique function $f(W) = \max_{e \in W^2} f(e)$.

For any binary character $\mathcal{X}$, there are exactly 2 vertices corresponding to $\mathcal{X}$ in int($\mathfrak{C}$). Therefore, if all characters are binary, there is a one-to-one correspondence between characters and potential fill edges e with $f(e) = 1$.

Corollary 2 ([55]). *Let $\mathfrak{C}$ be a set of binary characters. There exists a compatible subset $\mathfrak{C}' \subseteq \mathfrak{C}$ of size $|\mathfrak{C}'|$ if and only if there exists a triangulation $H \in Tr(int(\mathfrak{C}))$ with*

$$\sum_{e \in E(H) \setminus E(int(\mathfrak{C}))} f(e) = |\mathfrak{C} \setminus \mathfrak{C}'|.$$

By Corollary 2 the maximum compatibility problem of binary characters can be formulated as finding an optimal triangulation with respect to a fill-in-type cost function with a fill-edge function $f(e)$.

The formulation of binary maximum compatibility does not work for multi-state characters. Intuitively, this is because there can be more than one possible fill-edge that causes a character to not be compatible, resulting in the fact that the number of incompatible characters cannot be counted locally. However, it is clear that also in the multi-state case there is a minimal triangulation of the PI-graph that corresponds to a maximum size compatible subset of characters. We will use this fact in Section 4.4 to design a hybrid BT-MaxSAT algorithm for the maximum compatibility problem of multi-state characters.

3.6 Treelength

The length of a tree decomposition is the longest distance between two vertices in a same bag of the decomposition [34]. The treelength of a graph is the minimum length of a tree decomposition of the graph. Treelength was introduced by Dourisboure and Gavoille in 2007, motivated by applications in distance labeling and compact routing [2, 34]. In graph-theoretical context, bounded treelength graphs generalize chordal graphs in the sense that chordal graphs are exactly the graphs with treelength 1, and prior generalizations of the class of chordal graphs are known to have bounded treelength [34].

While deciding if a graph has treelength 1 is simply checking if the graph is chordal and thus linear-time, deciding if a graph has treelength $\leq k$ is NP-complete for every constant k at least 2 [80]. To the best of our knowledge, the only exact algorithm for computing treelength is an application of the BT algorithm presented by Lokshtanov [80].

Proposition 14 ([34]). *The treelength of a graph G is*

$$\min_{H \in Tr(G)} \max_{W \in MC(H)} \max_{u,v \in W} dist_G(u, v),$$

where $dist_G(u, v)$ is the distance between vertices u and v in the graph G.

The function $\max_{u,v \in W} \text{dist}_G(u, v)$ is monotone with respect to set inclusion because adding a vertex to W does not remove any existing pair $u, v \in W$. Therefore treelength can be formulated as finding an optimal triangulation with respect to a clique-max-type cost function with a clique function $f(W) = \max_{u,v \in W} \text{dist}_G(u, v)$.

Remark 2. *A graph parameter called treebreadth, which is closely related to treelength, has also been investigated in the literature [2, 35]. The definition of treebreadth is similar to treelength except that the clique function is the radius of the bag, i.e., $f(W) =$*

$\min_{v \in V(G)} \max_{u \in W} dist_G(u, v)$ *[35]. The radius of the bag is monotone with respect to set inclusion, and therefore treebreadth can be formulated as finding an optimal triangulation with respect to a clique-max-type cost function. While not done in this thesis, we note that the BT algorithm could similarly be evaluated on computing treebreadth. We focus on treelength because treebreadth and treelength are algorithmically similar in the context of the BT algorithm, and an optimal treelength decomposition is a 2-approximation of an optimal treebreadth decomposition [35].*

4 The Bouchitté–Todinca Algorithm

In 2001 Bouchitté and Todinca proposed an algorithm for computing treewidth and minimum fill-in in polynomial time in both the number of vertices and the number of minimal separators of the input graph [22, 23]. While the BT algorithm has been generalized and modified in multiple directions [16, 43, 44, 45, 86], all variants of the algorithm first enumerate special objects called potential maximal cliques (PMCs) of the input graph, and then use dynamic programming on top of them to find an optimal minimal triangulation of the graph. We refer to this two-phase approach as the BT algorithm.

In this chapter we overview the BT algorithm. We first review the central to BT definitions of minimal separators and potential maximal cliques and their properties connecting them to minimal triangulations. Then we present the PMC enumeration phase, which we refer to as PMC-ENUM, and the dynamic programming phase, which we refer to as BT-DP. Lastly, we present our MaxSAT adaptation of the BT-DP phase for the maximum compatibility problem of multi-state phylogenetic characters.

The PMC-ENUM algorithm that we use in this thesis is a novel modification of the original PMC-ENUM algorithm of Bouchitté and Todinca [22]. The modification is designed with the goal to allow an efficient practical implementation. As a subroutine, PMC-ENUM enumerates all minimal separators of the graph, for which we use Berry's algorithm [12]. The version of the BT-DP algorithm that we use in this thesis was introduced for treewidth and minimum fill-in by Fomin et al. [42], and later generalized to other cost functions by multiple authors [43, 45, 55, 80, 86]. The MaxSAT adaptation of BT-DP for the maximum compatibility problem of multi-state phylogenetic characters was introduced by us in [72].

4.1 Combinatorial Objects

Minimal separators and potential maximal cliques are the two central combinatorial objects considered in the BT algorithm, both closely connected to minimal triangulations.

4.1.1 Minimal Separators

A set of vertices $S \subseteq V(G)$ is an a,b-separator of a graph G if the vertices a and b are in different connected components of $G \setminus S$. In other words, all paths between a and b go through S. The set S is a minimal a,b-separator of G if no subset of S is also an a,b-separator [23].

Definition 12 ([23])**.** *Let G be a graph. A set of vertices $S \subseteq V(G)$ is a minimal separator of G if it is a minimal a,b-separator for some pair $a, b \in V(G)$.*

The set of minimal separators of a graph G is denoted by $\Delta(G)$.

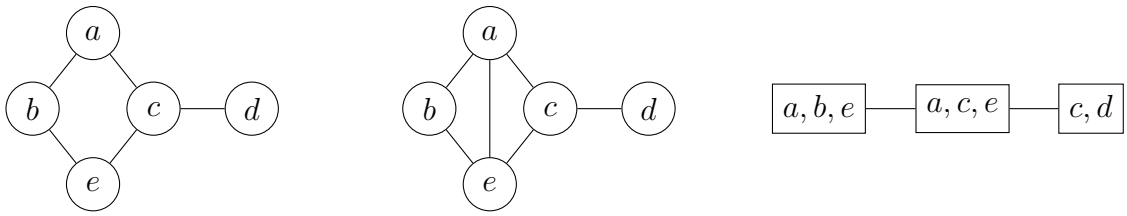

Figure 4.1: A graph (left), one of its minimal triangulations (middle), and a tree decomposition corresponding to the minimal triangulation (right).

Example 17. *Consider the graph G in Figure 4.1 (left). The set $\{a, e\}$ is a b,d-separator in G. It is a minimal b,d-separator because neither $\{a\}$ or $\{e\}$ is a b,d-separator, and therefore $\{a, e\} \in \Delta(G)$. Other minimal separators of G are $\{b, c\}$ and $\{c\}$. Note that a minimal separator can be a subset of another minimal separator.*

The concept of *full component* is used in multiple parts of the BT algorithm.

Definition 13 ([23])**.** *Let G be a graph and $X \subseteq V(G)$. A component $C \in \mathcal{C}(G \setminus X)$ is a full component of X in G if $N(C) = X$.*

Full components provide an alternative characterization of minimal separators.

Lemma 1 ([23])**.** *Let G be a graph and $a, b \in V(G)$. A set of vertices $S \subseteq V(G)$ is a minimal a,b-separator if and only if a and b are in different full components of S.*

Proof. A vertex $v \in S$ can be removed from an a,b-separator S while maintaining that S is an a,b-separator if and only if $v \notin N(C_a)$ or $v \notin N(C_b)$, where C_a is the component of $G \setminus S$ containing a and C_b is the component of $G \setminus S$ containing b. □

By Lemma 1, a vertex set is a minimal separator if and only if it has at least 2 full components.

Example 18. *Consider the graph G in Figure 4.1 (left). The vertex set $\{b, c\}$ is a minimal separator of G. Of the components $\mathcal{C}(G \setminus \{b, c\}) = \{\{a\}, \{d\}, \{e\}\}$, the components $\{a\}$ and $\{e\}$ are full components of $\{b, c\}$.*

Minimal separators of chordal graphs have a special structure that is related to their maximal cliques. Let $(T, \mathcal{B})$ be a tree decomposition of a chordal graph H such that $\mathcal{B} = \mathrm{MC}(H)$. The minimal separators of H are exactly the intersections of adjacent bags of $(T, \mathcal{B})$, i.e., $S \in \Delta(H)$ if and only if $S = B_i \cap B_j$ for some $(i, j) \in E(T)$ [23]. Therefore, every minimal separator of a chordal graph is a clique, and a chordal graph has at most $n - 1$ minimal separators.

The connection between minimal triangulations and minimal separators is even stronger.

Proposition 15 ([23])**.** *If G is a graph and H is a minimal triangulation of G, then $\Delta(H) \subseteq \Delta(G)$. Moreover, if $S \in \Delta(H)$, then $\mathcal{C}(H \setminus S) = \mathcal{C}(G \setminus S)$.*

By Proposition 15, if $(T, \mathcal{B})$ is a tree decomposition corresponding to a minimal triangulation H of a graph G, then all of the intersections of adjacent bags of $(T, \mathcal{B})$ are minimal separators of G. Moreover, any minimal separator of H separates the same pairs of vertices in H and G.

Example 19. *Consider the graph G, one of its minimal triangulations H, and a tree decomposition $(T, \mathcal{B})$ corresponding to H in Figure 4.1 (left, middle, and right, respectively). The minimal separators of H are $\{a, e\}$ and $\{c\}$, both of which are also minimal separators of G. The minimal separator $\{a, e\}$ is the intersection of the bags $\{a, b, e\}$ and $\{a, c, e\}$ of $(T, \mathcal{B})$, while the minimal separator $\{c\}$ is the intersection of the bags $\{a, c, e\}$ and $\{c, d\}$. Note also that $\mathcal{C}(H \setminus \{a, e\}) = \mathcal{C}(G \setminus \{a, e\}) = \{\{b\}, \{c, d\}\}$ and $\mathcal{C}(H \setminus \{c\}) = \mathcal{C}(G \setminus \{c\}) = \{\{a, b, e\}, \{d\}\}$.*

In addition to Proposition 15, it holds that for each minimal separator $S \in \Delta(G)$, there exists a minimal triangulation $H \in \mathrm{MT}(G)$ with $S \in \Delta(H)$ [23]. In other words, minimal separators of a graph are exactly the vertex sets that have potential to be minimal separators of a minimal triangulation of the graph.

4.1.2 Potential Maximal Cliques

Potential maximal cliques of a graph are the vertex sets that have potential to be maximal cliques of a minimal triangulation of the graph.

Definition 14 ([23])**.** *Let G be a graph. A vertex set $\Omega \subseteq V(G)$ is a potential maximal clique of G if there is a minimal triangulation $H \in MT(G)$ with $\Omega \in MC(H)$.*

The set of the PMCs of a graph G is denoted by $\Pi(G)$.

Example 20. *Consider the graph G in Figure 4.2 (left). The vertex set $\{b, c, e\}$ is a PMC of G because it is a maximal clique in the minimal triangulation H of G shown in Figure 4.2 (right). Other examples of PMCs of G include $\{a, b, c\}$, $\{b, d, e\}$, $\{c, e, f\}$ and $\{d, e, f\}$.*

Next we consider two crucial properties connecting the structure of PMCs and minimal separators. The following lemma formalizes how a PMC of a graph interacts with the rest of the graph only via the minimal separators it contains.

Figure 4.2: A graph with one of its potential maximal cliques colored gray (left) and one of the graph's minimal triangulations (right).

Lemma 2 ([23]). *If Ω is a potential maximal clique of a graph G, then for each component $C \in \mathcal{C}(G \setminus \Omega)$, the neighborhood $N(C) = S$ is a minimal separator of G. There are no other minimal separators $S \subseteq \Omega$.*

Note that such a component C is a full component of S. By Lemma 2, a PMC Ω of a graph G contains exactly $|\mathcal{C}(G \setminus \Omega)|$ minimal separators. We note that a minimal separator is never a PMC nor a superset of a PMC [23].

Example 21. *Consider the graph G in Figure 4.2 (left) and the PMC $\Omega = \{b, c, e\}$ of G. The minimal separators of G that are contained in Ω are $\{b, c\}$, $\{b, e\}$ and $\{c, e\}$. Each of them is a neighborhood of one of the components $\mathcal{C}(G \setminus \Omega) = \{\{a\}, \{d\}, \{f\}\}$. Note that these minimal separators are also minimal separators of the minimal triangulation H of G shown in Figure 4.2 (right) whose maximal clique Ω is.*

Lemma 3 ([23]). *Let G be a graph, S a minimal separator of G, and Ω a potential maximal clique of G with $S \subseteq \Omega$. There is a full component C of S with $\Omega \subseteq S \cup C$.*

Note that by disjointness of components, such a component C is unique. By combining Lemmas 2 and 3, we can deduce that each minimal separator S contained in a PMC Ω is a minimal a,b-separator for a pair a, b of vertices $a \in C_a$ and $b \in \Omega$, where $C_a \in \mathcal{C}(G \setminus \Omega)$.

Example 22. *Consider the graph G in Figure 4.2 (left) and the PMC $\Omega = \{b, c, e\}$ of G. The minimal separator $\{b, c\}$ of G is contained in Ω. Its full components are $\{a\}$ and $\{d, e, f\}$, with $\Omega \subseteq (\{b, c\} \cup \{d, e, f\})$.*

4.2 Enumeration Phase

The first phase of the BT algorithm is to enumerate all potential maximal cliques of the input graph [23]. Concurrently with the BT-DP algorithm, Bouchitté and Todinca introduced a minimal separator based PMC enumeration algorithm having time complexity $O(|\Delta(G)|^2 n^2 m)$, where $\Delta(G)$ is the set of minimal separators of the input graph G [22].

In this section we describe PMC-ENUM, which is a modification of the original algorithm of Bouchitté and Todinca for enumerating all PMCs of a graph [22]. Our modification reduces the time complexity slightly to $O(|\Delta(G)|^2 nm + |\Delta(G)| n^2 m)$. More importantly, the modification introduces additional insights that allow practical optimizations that are effective on typical inputs. PMC-ENUM first enumerates minimal separators with Berry's algorithm [12] and then generates potential maximal cliques from the minimal separators.

4.2.1 Enumerating Minimal Separators

We start by describing Berry's algorithm [12] for enumerating all minimal separators of a graph G in $O(|\Delta(G)| n^3)$ time. The algorithm first enumerates the minimal separators that

are contained in the neighborhood of some vertex and then generates all other minimal separators from this basis.

A minimal separator S is *close* to a vertex v if $S \subseteq N(v)$. Recall that the notation $N[v]$ denotes the closed neighborhood $N(v) \cup \{v\}$ of a vertex v.

Proposition 16 ([12]). *Let S be a minimal separator of a graph G and $v \in V(G)$. We have that $S \subseteq N(v)$ if and only if there is a component $C \in \mathcal{C}(G \setminus N[v])$ with $S = N(C)$.*

By Proposition 16, there are at most n^2 minimal separators that are close to a vertex, and they can be listed in $O(n^3)$ time. The other minimal separators can be generated from this basis via a production rule.

Definition 15 ([12]). *Let S be a minimal separator of a graph G. The set of minimal separators close to S in G is $\mathcal{R}(S) = \{N(C) \mid C \in \mathcal{C}(G \setminus (S \cup N(v))) \mid v \in S\}$.*

Definition 15 provides a production rule $\mathcal{R}$ to generate minimal separators $\mathcal{R}(S)$ that are close to a given minimal separator S. The rule considers each vertex $v \in S$ and computes the connected components of the graph $G \setminus (S \cup N(v))$. The neighborhoods of the components are the resulting minimal separators. In this process, at most $|S|n$ minimal separators are generated from a minimal separator S. The rule can be implemented to compute the set $\mathcal{R}(S)$ in $O(|S|n^2)$ time.

Proposition 17 ([12]). *If S is a minimal separator of a graph G, then every $S' \in \mathcal{R}(S)$ is also a minimal separator of G. All minimal separators of G can be generated by successively applying the production rule $\mathcal{R}(S)$, starting from the minimal separators that are close to a vertex.*

Algorithm 1: MS-ENUM

Input : A graph G.
Output: The minimal separators of G.

```
1  Let Δ be an empty set of minimal separators.
2  for v ∈ V(G) do
3  |  for C ∈ C(G \ N[v]) do
4  |  |  Δ.insert(N(C))
5  Let Q be an empty queue of minimal separators.
6  Q.push(Δ_1, ..., Δ_|Δ|)
7  while Q not empty do
8  |  S ← Q.pop()
9  |  for S' ∈ R(S) do
10 |  |  if S' ∉ Δ then
11 |  |  |  Δ.insert(S')
12 |  |  |  Q.push(S')
13 return Δ
```

Algorithm 1 presents Berry's algorithm for enumerating minimal separators [12] as pseudocode, which we will refer to as MS-ENUM. On lines 2 to 4 MS-ENUM enumerates the basis, the minimal separators that are close to a vertex. Then on lines 7 to 12 MS-ENUM considers each enumerated minimal separator and applies the production rule to it to generate more minimal separators.

Theorem 1 ([12]). *MS-ENUM enumerates $\Delta(G)$ in $O(|\Delta(G)|n^3)$ time for any graph G.*

Theorem 1 follows from Proposition 17 and from the fact that MS-ENUM computes the basis in $O(n^3)$ time and applies the production rule in $O(n^3)$ time per minimal separator.

4.2.2 Enumerating Potential Maximal Cliques

We describe PMC-ENUM, our modification of the algorithm of Bouchitté and Todinca [22] for enumerating all potential maximal cliques of a graph.

PMC-ENUM first enumerates minimal separators with MS-ENUM, and then generates the set of PMCs $\Pi(G)$ of the input graph G in three steps. The first step works via induction using $\Pi(G \setminus \{a\})$, where $a \in V(G)$. The second step enumerates the PMCs that are not enumerated by the first step and contain the vertex a. The third step enumerates the remaining PMCs. The second and the third step make use of the minimal separators of G, generating candidates of PMCs from minimal separators. All of the candidates have to be checked for being actual PMCs, for which we make use of the following proposition.

Proposition 18 ([22]). *Let G be a graph. A vertex set $\Omega \subseteq V(G)$ is a potential maximal clique of G if and only if*

1. *for any pair of distinct vertices $u, v \in \Omega$, either $\{u, v\} \in E(G)$ or there is a component $C \in \mathcal{C}(G \setminus \Omega)$ with $\{u, v\} \subseteq N(C)$, and*
2. *no component of Ω is full, i.e., $N(C) \subset \Omega$ for all $C \in \mathcal{C}(G \setminus \Omega)$.*

We will refer to condition 1 of Proposition 18 as the *cliquish condition*, following [99], and to condition 2 of Proposition 18 as the *no full component condition*. The cliquish condition can be equivalently stated as follows: for each pair of distinct vertices $u, v \in \Omega$, there is a path $u, w_1, \ldots, w_p, v$ in G whose intermediate vertices $w_1, \ldots, w_p$ are in $V(G) \setminus \Omega$. We call this kind of path a path *outside* of Ω.

The cliquish condition can be checked by a graph search from each of the vertices of Ω and the no full component condition with a single graph search finding the neighborhoods of all components of $G \setminus \Omega$. The time complexities of these checks are $O(n + |\Omega|m)$ and $O(n + m)$, respectively.

Corollary 3 ([22]). *Given a graph G and a vertex set $\Omega \subseteq V(G)$, it can be decided in $O(nm)$ time if Ω is a potential maximal clique of G.*

Induction on Induced Subgraphs. The first step of PMC-ENUM works by induction, enumerating PMCs of G based on PMCs of an induced subgraph $G \setminus \{a\}$, for an arbitrary selected vertex $a \in V(G)$. In order for this induction to make sense, minimal separators and PMCs need to behave well with respect to induced subgraphs. The next two lemmas formalize this.

Lemma 4 ([22])**.** *Let G be a graph with $a, x, y \in V(G)$. If S is a minimal x,y-separator in $G \setminus \{a\}$, then either S or $S \cup \{a\}$ is a minimal x,y-separator in G.*

Proof. If S is not x,y-separator in G, then $S \cup \{a\}$ is. The separator $S \cup \{a\}$ is minimal because it has the same full components in G that S has in $G \setminus \{a\}$. □

Lemma 4 implies important facts for PMC-ENUM. First, by Lemma 4 the number of minimal separators of $G \setminus \{a\}$ is at most the number of minimal separators of G. Second, all minimal separators of $G \setminus \{a\}$ can be computed from the minimal separators of G. In particular, by testing if $S \setminus \{a\}$ is a minimal separator of $G \setminus \{a\}$ for each $S \in \Delta(G)$, the set $\Delta(G \setminus \{a\})$ can be enumerated in $O(|\Delta(G)|(n+m))$ time given $\Delta(G)$.

Lemma 5. *Let G be a graph, $a \in V(G)$, and $\Omega \in \Pi(G \setminus \{a\})$. Exactly one of Ω and $\Omega \cup \{a\}$ is a potential maximal clique of G.*[1]

Proof. First, note that Ω satisfies the cliquish condition in G and $\Omega \cup \{a\}$ satisfies the no full component condition in G. Now, if there is a path in G outside of Ω from the vertex a to every vertex $v \in \Omega$, then $\Omega \cup \{a\}$ satisfies the cliquish condition, and Ω has a full component containing a. If there is no such path, then $\Omega \cup \{a\}$ does not satisfy the cliquish condition, but Ω has no full components. □

Lemma 5 implies that the number of PMCs of $G \setminus \{a\}$ is at most the number of PMCs of G. Following Lemma 5, the first step of PMC-ENUM is to call PMC-ENUM recursively on $G \setminus \{a\}$ to enumerate $\Pi(G \setminus \{a\})$ and to check for each $\Omega \in \Pi(G \setminus \{a\})$ if Ω or $\Omega \cup \{a\}$ is a PMC of G. Note that by the proof of Lemma 5, this check can be implemented in linear time per PMC because it is sufficient to check if Ω has a full component in G.

The Second Step of PMC-ENUM. Next we characterize the PMCs that are not enumerated by the first step and contain the vertex a. These PMCs are enumerated by the second step of PMC-ENUM.

Lemma 6 ([22])**.** *Let Ω be a potential maximal clique of a connected graph G with $|V(G)| \geq 2$. If $a \in \Omega$ and $\Omega \setminus \{a\} \notin \Pi(G \setminus \{a\})$, then $\Omega \setminus \{a\} \in \Delta(G)$.*

Proof. Note that the components of the graph $G \setminus \Omega$ are the same as the components of the graph $(G \setminus \{a\}) \setminus (\Omega \setminus \{a\})$. Therefore $\Omega \setminus \{a\}$ does not violate the cliquish condition in $G \setminus \{a\}$, and so it has a full component C in $G \setminus \{a\}$. Because Ω has no full component

[1] A weaker version of the lemma is presented in [22].

in G, it holds that $N(C) = \Omega \setminus \{a\}$ in G. Because $C \in \mathcal{C}(G \setminus \Omega)$, it follows from Lemma 2 that $N(C)$ is a minimal separator of G. □

By Lemma 6, the PMCs that are not enumerated by the first step and contain the vertex a can be constructed from minimal separators. Based on Lemma 6, the second step of PMC-ENUM is to check for each minimal separator $S \in \Delta(G)$ if $S \cup \{a\}$ is a PMC. This together with the first step is sufficient to find all PMCs of G that contain the vertex a.

Note that Lemma 6 requires the graph G to be connected and have at least two vertices. We assume that the input graph is connected and that in each recursion step we choose $a \in V(G)$ so that the graph $G \setminus \{a\}$ is connected. The graph with only one vertex is the base case, containing exactly one PMC and handled separately.

The Third Step of PMC-ENUM. The remaining case covers the PMCs that do not contain the vertex a and cannot be constructed with Lemma 5 from the PMCs of $G \setminus \{a\}$. Our characterization of these PMCs is a novel adaptation of the results of Bouchitté and Todinca, based on the following lemma from [22], which reveals the structure of the PMCs that contain so-called *active separators*.

Lemma 7 ([22])**.** *Let G be a graph, $\Omega \in \Pi(G)$, $S \in \Delta(G)$, and $S \subseteq \Omega$. Let C_Ω denote the full component of S with $\Omega \subseteq S \cup C_\Omega$. If there is a pair of vertices $x, y \in S$ such that $\{x, y\} \notin E(G)$ and S is the only minimal separator contained in Ω that contains x and y, then $\Omega \setminus S$ is a minimal x,y-separator in $G[C_\Omega \cup \{x, y\}]$.*

A minimal separator S containing x and y as in Lemma 7 is called an active separator of the PMC Ω [22]. We use Lemma 7 to prove the following theorem, which characterizes the PMCs not characterized by Lemmas 5 and 6. Figure 4.3 illustrates the objects used in the proof, namely the active separator S, its full components C_a and C_Ω, vertices a, x, y, and the minimal x,y-separator T.

Theorem 2. *Let G be a graph and $a \in V(G)$. If $\Omega \in \Pi(G) \setminus \Pi(G \setminus \{a\})$ and $a \notin \Omega$, then $\Omega = S \cup T \setminus \{a\}$ for some $S, T \in \Delta(G)$. Moreover, $a \notin S$, $S \notin \Delta(G \setminus \{a\})$, and $a \in T$.*

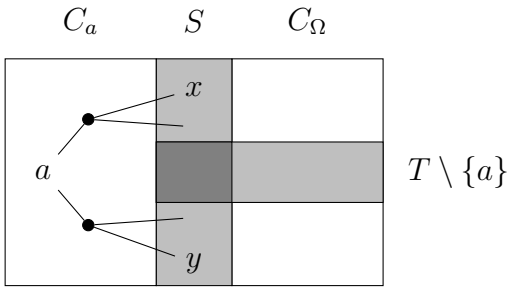

Figure 4.3: Illustration of the proof of Theorem 2. The vertex sets S and $T \setminus \{a\}$ whose union forms the potential maximal clique Ω are colored gray.

Proof. If Ω has no full components in G, then it also does not have full components in $G \setminus \{a\}$. Thus the reason why Ω is not a PMC of $G \setminus \{a\}$ is that it violates the cliquish condition, i.e., there is a pair of vertices $x, y \in \Omega$ so that a is on all paths between x and y that are outside of Ω. Let C_a be the component of $G \setminus \Omega$ that contains a. It follows from Lemma 2 that $S = N(C_a)$ is a minimal separator of G. Furthermore, $x, y \in S$ because the path between x and y in G goes through a.

Let C_Ω be the full component of S that contains $\Omega \setminus S$. By Lemma 3, the component C_Ω exists and is unique. The only components $C \in \mathcal{C}(G \setminus S)$ with $x, y \in N(C)$ are C_a and C_Ω, since otherwise there would be a path between x and y outside of Ω in $G \setminus \{a\}$. Therefore C_a and C_Ω are the only full components of S. Since the vertex a separates x from y in $G[C_a \cup \{x, y\}]$, the only component of $G \setminus \{a\} \setminus S$ that has $\{x, y\}$ in its neighborhood is C_Ω, and therefore $S \notin \Delta(G \setminus \{a\})$.

Note that the minimal separator S is an active separator of Ω whose properties are summarized in Lemma 7. In particular, it follows that the set $\Omega \setminus S$ is a minimal x,y-separator in $G[C_\Omega \cup \{x, y\}]$. Next we gradually add vertices to the minimal x,y-separator $\Omega \setminus S$ and to the induced subgraph $G[C_\Omega \cup \{x, y\}]$ to finally yield a minimal x,y-separator T in the graph G.

First, the set $\Omega \setminus S \cup \{a\}$ is a minimal x,y-separator in $G[C_\Omega \cup \{x, y\} \cup C_a]$. This is because there are no edges between C_a and C_Ω and all paths between x and y in $G[\{x, y\} \cup C_a]$ go through a. Adding the other components of $G \setminus S$ to the induced subgraph, we deduce that $\Omega \setminus S \cup \{a\}$ is also a minimal x,y-separator in the graph $G[(V(G) \setminus S) \cup \{x, y\}]$ because the other components do not neighbor any other vertices of $G[C_\Omega \cup \{x, y\} \cup C_a]$ than at most one of x or y. Finally, by applying Lemma 4 to add the vertices $S \setminus \{x, y\}$ to the induced subgraph and some of them to the minimal x,y-separator, it follows that there is a minimal x,y-separator $T \in \Delta(G)$ such that $T \setminus S = \Omega \cup \{a\} \setminus S$, and therefore $T \cup S \setminus \{a\} = \Omega$. □

By Theorem 2, the PMCs of G that cannot be constructed from the PMCs of $G \setminus \{a\}$ and do not contain a can be constructed by considering pairs of minimal separators $S, T \in \Delta(G)$. Furthermore, there are additional conditions to limit the number of pairs S, T to consider. Corollary 4 summarizes the steps of PMC-ENUM to characterize all PMCs of a graph.

Corollary 4. *Let Ω be a potential maximal clique of a connected graph G and a a vertex of G. If $|V(G)| \geq 2$, one of the following holds.*

1. $\Omega \setminus \{a\} \in \Pi(G \setminus \{a\})$.
2. $\Omega \setminus \{a\} \in \Delta(G)$.
3. $\Omega = S \cup T \setminus \{a\}$, *where* $S, T \in \Delta(G)$, $a \notin S$, $S \notin \Delta(G \setminus \{a\})$, *and* $a \in T$.

Proof. By Lemma 6, if $a \in \Omega$ and (1) does not hold, then (2) holds. By Theorem 2, if $a \notin \Omega$ and (1) does not hold, then (3) holds. □

Algorithm 2: PMC-ENUM

Input : A connected graph G.
Output: The potential maximal cliques of G.

```
1  Let Π be an empty set of potential maximal cliques.
2  Δ ← MS-ENUM(G)
3  O[1...n] ← VERTEX-ORDER(G)
4  G_1 ← G
5  for i ← 1 to n − 1 do
6  |  Π_i ← ∅
7  |  a ← O[i]
8  |  Δ_T ← {S \ {a} | a ∈ S ∈ Δ}
9  |  Δ_N ← {S ∈ Δ \ Δ_T | ISMINSEP(S, G_i \ {a})}
10 |  Δ_S ← Δ \ Δ_T \ Δ_N
11 |  for S ∈ Δ_N ∪ Δ_S do
12 |  |  if ISPMC(S ∪ {a}, G_i) then
13 |  |  |  Π_i ← Π_i ∪ {S ∪ {a}}
14 |  Π_i ← Π_i ∪ COMBINE(Δ_S, Δ_T, G_i)
15 |  Π ← Π ∪ EXTEND(Π_i, O[i − 1...1], G)
16 |  Δ ← Δ_T ∪ Δ_N
17 |  G_{i+1} ← G_i \ {a}
18 Π ← Π ∪ EXTEND({{O[n]}}, O[n − 1...1], G)
19 return Π
```

PMC-ENUM is an implementation of the recursive procedure suggested by Corollary 4. PMC-ENUM is presented in pseudocode as Algorithm 2. Even though the principle underlying PMC-ENUM is induction over induced subgraphs, the pseudocode is not recursive in order to better illustrate the running time of the algorithm and to describe our actual implementation of it.

PMC-ENUM takes as input a graph G. It first computes the set Δ of minimal separators of G with Berry's algorithm and uses a subroutine VERTEX-ORDER to find an order $O[1 \dots n]$ to remove vertices from G so that the resulting graph is always connected. The successive induced subgraphs resulting from removing vertices from G are $G_1 = G$, $G_2 = G \setminus \{O[1]\}$, $G_3 = G \setminus \{O[1], O[2]\}, \dots, G_n = G[\{O[n]\}]$.

The graphs $G_1, \dots, G_{n-1}$ are considered in the loop consisting of lines 5 to 17. First, on lines 6 to 14 the subset Π_i of the PMCs of G_i is computed, containing the PMCs that correspond to cases 2 and 3 of Corollary 4. On line 15 each PMC $\Omega' \in \Pi_i$ is used to construct a PMC Ω of G with $\Omega' \subseteq \Omega$ by successively applying Lemma 5 with the EXTEND subroutine. On lines 16 and 17 the next graph G_{i+1} and its minimal separators are computed. Line 18 handles the base case of a graph with a single vertex.

In more detail, the PMCs corresponding to cases 2 and 3 of Corollary 4 are computed by first partitioning the set of minimal separators of G_i into three parts, Δ_S, Δ_T, and

Δ_N. The sets Δ_S and Δ_T correspond to the minimal separators S and T of case 3 of Corollary 4 and Δ_N to the rest of the minimal separators of G_i. In particular, Δ_T is the set of minimal separators of G_i that contain the vertex a, with the vertex a removed from each minimal separator in Δ_T, and Δ_S is the set of minimal separators of G_i that do not contain a and are not minimal separators of G_{i+1}. The PMCs that correspond to case 2 are computed on lines 11 to 13, and the PMCs that correspond to case 3 are computed with the COMBINE$(\Delta_S, \Delta_T, G_i)$ subroutine which tries all pairs $S \in \Delta_S$ and $T \in \Delta_T$ to form a PMC $S \cup T$ of G_i.

Theorem 3. *Any graph G has at most $|\Delta(G)|^2 + n|\Delta(G)| + 1$ potential maximal cliques and PMC-ENUM enumerates them in $O(|\Delta(G)|^2 nm + |\Delta(G)|n^2 m)$ time.*

Proof. The correctness of PMC-ENUM follows from Corollary 4. To bound the number of PMCs, note that there are at most $|\Delta(G)|n$ PMC candidates corresponding to case 2 of Corollary 4. For case 3 of Corollary 4, note that the sum of the cardinalities of Δ_S across all iterations is $|\Delta(G)|$, so in total at most $|\Delta(G)|^2$ candidates are considered. For the time complexity, recall that by Corollary 3, each PMC candidate can be checked in $O(nm)$ time. EXTEND can be implemented in a total of $O(nm)$ time per PMC because only the no full component condition of a PMC needs to be checked when determining whether Ω or $\Omega \cup \{a\}$ is a PMC. □

Theorem 3 improves the results of Bouchitté and Todinca [22] by a factor of n in the leading term in both the number of PMCs and the running time to enumerate them. The more important property of our modification that differs from the original algorithm [22] is that the COMBINE$(\Delta_S, \Delta_T, G_i)$ subroutine produces PMCs of G_i that are supersets of the minimal separators given in the sets Δ_S and Δ_T. As we will explain in more detail in Section 5.3, this allows to prune the sets Δ_S and Δ_T based on problem-specific insights that restrict the set of PMCs that need to be enumerated.

4.3 Dynamic Programming Phase

In this section we describe BT-DP, the dynamic programming phase of the BT algorithm. BT-DP takes as an input a graph, the set of potential maximal cliques of the graph, and a description of a cost function on triangulations of the graph. BT-DP outputs the cost of an optimal minimal triangulation with respect to the cost function as well as a corresponding triangulation. Our presentation of BT-DP is based on [23, 42, 43, 45].

4.3.1 Characterization of Minimal Triangulations

In BT-DP, the subproblems of the dynamic programming correspond to *blocks* of the input graph.

Definition 16 ([23])**.** *Let G be a graph. A pair (S, C) is a block of G if S is a minimal separator of G and C is a full component of S in G.*

More accurately, BT-DP computes an optimal minimal triangulation for each *realization* of a block of the input graph.

Definition 17 ([23])**.** *Let G be a graph and (S, C) a block of G. The realization of (S, C), denoted by $R(S, C)$, is the graph with the vertex set $V(R(S, C)) = S \cup C$ and the edge set $E(R(S, C)) = S^2 \cup E(G[S \cup C])$.*

In words, the realization of (S, C) is the induced subgraph $G[S \cup C]$ with the minimal separator S completed into a clique. To simplify presentation, following [43], we let the pair $(\emptyset, V(G))$ also be a block of any graph G. Now the realization $R(\emptyset, V(G)) = G$ corresponds to the root state of the dynamic programming. All properties of blocks presented in this section also hold for the special block $(\emptyset, V(G))$.

Example 23. *Consider the graph G in Figure 4.4 (left). The pair of sets $(S, C) = (\{b, c\}, \{d, e, f, g, h\})$ is a block of G because S is a minimal separator of G and C is a full component of S in G. The realization $R(S, C)$ is shown in Figure 4.4 (middle left). It consists of the induced subgraph $G[S \cup C]$ with the added edge $\{b, c\}$.*

BT-DP computes an optimal triangulation of a realization $R(S, C)$ based on optimal triangulations of realizations $R(S_i, C_i)$ with $(S_i \cup C_i) \subset (S \cup C)$. The transition to assemble minimal triangulations of smaller realizations into a minimal triangulation of a larger realization makes use of potential maximal cliques. To present the transition, we need to review properties connecting the structure of PMCs and blocks.

Recall that by Lemma 2, for a PMC Ω of a graph G, each component $C \in \mathcal{C}(G \setminus \Omega)$ is a full component of a minimal separator $N(C) = S$. Furthermore, the pair (S, C) is a block.

Definition 18 ([23])**.** *Let G be a graph and Ω a potential maximal clique of G. The associated blocks of Ω in G are $\mathcal{A}_G(\Omega) = \{(N(C), C) \mid C \in \mathcal{C}(G \setminus \Omega)\}$.*

Example 24. *Consider the graph G in Figure 4.4 (left). The set $\Omega = \{b, c, e, h\}$ is a PMC of G. The associated blocks of Ω in G are $(\{b, c\}, \{a\})$, $(\{b, h\}, \{d, g\})$ and $(\{c, h\}, \{f\})$.*

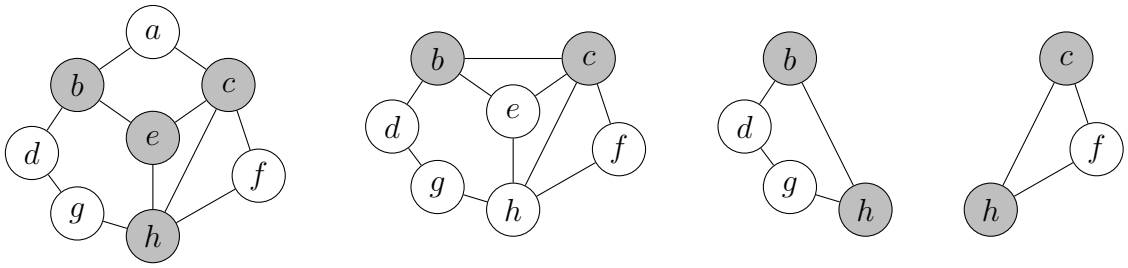

Figure 4.4: A graph with one of its potential maximal cliques colored gray (left) and three realizations $R(S, C)$ of blocks (S, C) of the graph with the minimal separators S colored gray.

The intuition of the BT algorithm is that if we choose a PMC Ω to be a maximal clique of a minimal triangulation of a graph G, then we can build the rest of the minimal triangulation by choosing an arbitrary minimal triangulation for each realization of a block $(S_i, C_i) \in \mathcal{A}_G(\Omega)$. In order for this kind of a recursive characterization to be computationally feasible, it should be easy to handle PMCs and blocks of realizations $R(S, C)$.

Lemma 8 ([23])**.** *Let (S, C) be a block of a graph G. If Ω is a potential maximal clique of $R(S, C)$ and $S \subseteq \Omega$, then $\Omega \in \Pi(G)$ and $\mathcal{A}_{R(S,C)}(\Omega) \subseteq \mathcal{A}_G(\Omega)$.*

By Lemma 8, if we consider only the PMCs that contain the minimal separator S of a realization $R(S, C)$, then handling the recursion is simple: all resulting blocks are also blocks of G and all of the PMCs of $R(S, C)$ are also PMCs of G. The restriction to PMCs containing S makes sense because S is a clique in $R(S, C)$, and therefore any triangulation of $R(S, C)$ has a maximal clique that contains S.

The PMCs Ω of the graph $R(S, C)$ with $S \subseteq \Omega$ are denoted by $\Pi(S, C)$. Recall that by Lemma 3, for any minimal separator S and PMC Ω with $S \subseteq \Omega$ there is exactly one full component C of S such that $\Omega \subseteq S \cup C$. Combining this with Lemma 8, it follows that $\Pi(S, C) = \{\Omega \in \Pi(G) \mid S \subseteq \Omega \subseteq S \cup C\}$ [23].

Example 25. *Consider the graph G in Figure 4.4 (left) and the block $(S, C) = (\{b, c\}, \{d, e, f, g, h\})$ of G. The realization $R(S, C)$ is shown in Figure 4.4 (middle left). The set $\Omega = \{b, c, e, h\}$ is a PMC of G. It is also a PMC of $R(S, C)$ with $S \subseteq \Omega$, and therefore $\Omega \in \Pi(S, C)$. The associated blocks of Ω in $R(S, C)$ are $\mathcal{A}_{R(S,C)}(\Omega) = \{(\{b, h\}, \{d, g\}), (\{c, h\}, \{f\})\}$. Note that $\mathcal{A}_{R(S,C)}(\Omega) \subseteq \mathcal{A}_G(\Omega)$.*

Now we can give the main theorem for characterizing the minimal triangulations of a graph.

Theorem 4 ([23, 42])**.** *Let G be a graph and (S, C) a block of G. The graph H is a minimal triangulation of $R(S, C)$ if and only if (i) $V(H) = S \cup C$ and (ii) there is a potential maximal clique $\Omega \in \Pi(S, C)$ such that*

$$E(H) = \Omega^2 \cup \bigcup_{(S_i, C_i) \in \mathcal{A}_{R(S,C)}(\Omega)} E(H_i),$$

where H_i is any minimal triangulation of $R(S_i, C_i)$.

The root of the recursion of Theorem 4 is given by the special block $R(\emptyset, V(G)) = G$. When a minimal triangulation is constructed with the recursion, each maximal clique of the minimal triangulation is considered exactly once as the PMC Ω. This allows adapting Theorem 4 to also compute the cost of the resulting minimal triangulation on the types of cost functions that we consider and to simultaneously optimize the choice of the PMC Ω to minimize the cost.

4.3.2 Finding Optimal Minimal Triangulations

Next we formulate the use of Theorem 4 to compute optimal triangulations with respect to the cost function types presented in Definitions 3–5 of Section 2.5, namely, clique-max-type, fill-in-type, and clique-sum-type. Note that even though we defined these cost functions as functions $C : \mathrm{Tr}(G) \to \mathbb{R}_{\geq 0}$ from the set $\mathrm{Tr}(G)$ of triangulations of an input graph G, they can be extended into functions that are defined for all triangulations of realizations of blocks of G.

Recall that computing treewidth is equivalent to finding a triangulation H that minimizes the cost function $C(H) = \max_{W \in \mathrm{MC}(H)} |W| - 1$ and therefore can be formulated as finding an optimal triangulation with respect to a clique-max-type cost function with a clique function $f(W) = |W| - 1$.

Corollary 5 (**Clique-max-type** [23, 42, 45])**.** *Let G be a graph, (S, C) a block of G, and $C(H)$ a clique-max-type cost function with a clique function f. The cost of an optimal triangulation of $R(S, C)$ with respect to $C(H)$ is*

$$C_{opt}(R(S,C)) = \min_{\Omega \in \Pi(S,C)} \max \left(f(\Omega), \max_{(S_i,C_i) \in \mathcal{A}_{R(S,C)}(\Omega)} C_{opt}(R(S_i,C_i)) \right).$$

The cost of an optimal triangulation of G is $C_{opt}(R(\emptyset, V(G)))$. Corollary 5 can be easily extended to also compute a corresponding optimal triangulation, as we will do in Algorithm 3.

Recall that computing the minimum fill-in is equivalent to finding a triangulation H of a graph G that minimizes the cost function $C(H) = |E(H) \setminus E(G)|$ and therefore can be formulated as finding an optimal triangulation with respect to a fill-in-type cost function with a fill-edge function $f(e) = 1$.

Corollary 6 (**Fill-in-type** [23, 42, 45])**.** *Let G be a graph, (S, C) a block of G, and $C(H)$ a fill-in-type cost function with a fill-edge function f. The cost of an optimal triangulation of $R(S, C)$ with respect to $C(H)$ is*

$$C_{opt}(R(S,C)) = \min_{\Omega \in \Pi(S,C)} \left(\sum_{e \in \Omega^2 \setminus E(R(S,C))} f(e) + \sum_{(S_i,C_i) \in \mathcal{A}_{R(S,C)}(\Omega)} C_{opt}(R(S_i,C_i)) \right).$$

Recall that in clique-sum-type cost functions the cost of a triangulation is defined as a sum over its maximal cliques.

Corollary 7 (**Clique-sum-type** [16, 23])**.** *Let G be a graph, (S, C) a block of G, and $C(H)$ a clique-sum-type cost function with a clique function f. The cost of an optimal triangulation of $R(S, C)$ with respect to $C(H)$ is*

$$C_{opt}(R(S,C)) = \min_{\Omega \in \Pi(S,C)} \left(f(\Omega) + \sum_{(S_i,C_i) \in \mathcal{A}_{R(S,C)}(\Omega)} C_{opt}(R(S_i,C_i)) \right).$$

4.3.3 The BT-DP Algorithm

Next we represent the BT-DP algorithm to compute optimal triangulations as formulated in Corollaries 5–7. We parameterize BT-DP with a binary operation $\oplus$ and a function g to represent the three types of cost functions in a unified manner.

Definition 19. *Let G be a graph and (S, C) a block of G. Furthermore, let $\oplus$ be either the maximum or the sum and $g : \Pi(G) \times (\Delta(G) \cup \{\emptyset\}) \to \mathbb{R}_{\geq 0}$ a function. The cost of an optimal minimal triangulation of $R(S, C)$ with respect to $\oplus$ and g is*

$$C_{opt}(R(S,C)) = \min_{\Omega \in \Pi(S,C)} \left(g(\Omega, S) \oplus \bigoplus_{(S_i,C_i) \in \mathcal{A}_{R(S,C)}(\Omega)} C_{opt}(R(S_i, C_i)) \right).$$

The formulas of Corollaries 5–7 are special cases of the formula of Definition 19. In particular, clique-max-type can be formulated as $\oplus = \max$ and $g(\Omega, S) = f(\Omega)$, fill-in-type as $\oplus = +$ and $g(\Omega, S) = \sum_{e \in (\Omega^2 \setminus S^2) \setminus E(G)} f(e)$, and clique-sum-type as $\oplus = +$ and $g(\Omega, S) = f(\Omega)$. Note that we do not claim that the formula of Definition 19 would compute anything sensible for arbitrary $\oplus$ and g.

Algorithm 3 presents BT-DP as pseudocode. BT-DP takes as an input a graph G, the set $\Pi(G)$ of its PMCs, and the functions $\oplus$ and g defining the cost function as formulated in Definition 19. BT-DP returns an optimal minimal triangulation of G with respect to $\oplus$ and g and the cost of the triangulation.

We divide BT-DP into three phases. The precomputation phase computes the blocks of G and the relations $\mathcal{A}_G(\Omega)$ and $\Pi(S, C)$ that link PMCs to blocks and blocks to PMCs. The dynamic programming phase computes the cost of an optimal minimal triangulation for each realization of a block by dynamic programming as suggested in Definition 19. The construction phase constructs an optimal triangulation of G based on tables computed by the dynamic programming phase.

The precomputation phase. BT-DP first iterates over all PMCs of G on lines 4 to 8, applying Lemma 2 to compute the blocks of G and the relation $\mathcal{A}$ mapping each PMC Ω to the set of its associated blocks $\mathcal{A}(\Omega)$. For each PMC Ω, the components of $G \setminus \Omega$ and their neighborhoods are computed and stored. There are at most $|\Pi(G)|n$ blocks and association relations and they are computed in $O(|\Pi(G)|n^2)$ time. On lines 9 to 14 BT-DP applies Lemma 3 to compute the relation $\Pi(S, C)$ mapping each block (S, C) to PMCs Ω with $S \subseteq \Omega \subseteq S \cup C$. This relation is computed similarly as $\mathcal{A}$ expect for the fact that a different full component of the minimal separator is considered. By similar arguments, the sets $\Pi(S, C)$ over all blocks (S, C) contain at most a total of $|\Pi(G)|n$ pointers and can be computed in $O(|\Pi(G)|nm)$ time.

The dynamic programming phase. On line 16 the blocks are sorted in a non-decreasing order of the number of vertices, so that the costs of all smaller blocks are computed

Algorithm 3: BT-DP

Input : A graph G, the set $\Pi(G)$ of its PMCs, and functions $\oplus$ and g.

Output: An optimal minimal triangulation of G with respect to $\oplus$ and g and the cost of the triangulation.

Precomputation Phase:
Let B be an empty set of blocks.
B.insert($(\emptyset, V(G))$)
for $\Omega \in \Pi(G)$ **do**
  Let $\mathcal{A}(\Omega)$ be a set of pointers to blocks.
  for $C \in \mathcal{C}(G \setminus \Omega)$ **do**
    B.insert($(N(C), C)$)
    $\mathcal{A}(\Omega)$.insert($(N(C), C)$)
For each $(S, C) \in B$, let $\Pi(S, C)$ be a set of pointers to PMCs.
for $\Omega \in \Pi(G)$ **do**
  $\Pi(\emptyset, V(G))$.insert(Ω)
  for $D \in \mathcal{C}(G \setminus \Omega)$ **do**
    Let C be the component of $G \setminus N(D)$ that intersects with Ω.
    $\Pi(N(C), C)$.insert(Ω)

Dynamic Programming Phase:
Sort B in non-decreasing order of $|S| + |C|$.
Let dp and $optChoice$ be tables on blocks B.
for $(S, C) \in B$ **do**
  $dp[(S, C)] = \infty$
  for $\Omega \in \Pi(S, C)$ **do**
    $cost \leftarrow g(\Omega, S)$
    for $(S_i, C_i) \in \mathcal{A}(\Omega)$ **do**
      if $C_i \subseteq C$ **then**
        $cost \leftarrow cost \oplus dp[(S_i, C_i)]$
    if $cost < dp[(S, C)]$ **then**
      $dp[(S, C)] \leftarrow cost$
      $optChoice[(S, C)] \leftarrow \Omega$

Construction Phase:
$H \leftarrow G$
Let Q be a queue.
Q.push($(\emptyset, V(G))$)
while Q *not empty* **do**
  $(S, C) \leftarrow Q$.pop()
  $\Omega \leftarrow optChoice[(S, C)]$
  $E(H) \leftarrow E(H) \cup \Omega^2$
  for $(S_i, C_i) \in \mathcal{A}(\Omega)$ **do**
    if $C_i \subseteq C$ **then**
      Q.push((S_i, C_i))
return $H, dp[(\emptyset, V(G))]$

before a larger block. Then, on lines 17 to 27 the cost of an optimal triangulation of the realization of each block (S, C) is computed. The computation first initializes the cost to be infinite, and then considers each PMC $\Omega \in \Pi(S, C)$ to find the optimal cost as suggested by Theorem 4. The cost of an optimal minimal triangulation of $R(S, C)$ is stored in $dp[(S, C)]$, and the corresponding PMC to choose to construct an optimal triangulation is stored in $optChoice[(S, C)]$. Finally, after $dp[(S, C)]$ has been computed for all blocks, the cost of an optimal triangulation of G is $dp[(\emptyset, V(G))]$.

The construction phase. On lines 29 to 38 BT-DP constructs an optimal minimal triangulation H of G. The construction is done in a breadth-first-search manner, making use of the *optChoice* table computed in the dynamic programming phase.

Proposition 19. *Given a graph G, the set $\Pi(G)$ of its potential maximal cliques, and functions $\oplus$ and g, where $\oplus$ is either the maximum or the sum and g is computable in $O(g_t)$ time, BT-DP finds an optimal minimal triangulation of G with respect to $\oplus$ and g in $O(|\Pi(G)|n(m + g_t))$ time.*

Proof. The blocks and their relations to PMCs can be computed in $O(|\Pi(G)|nm)$ time as discussed earlier. Because the total size of $\Pi(S, C)$ over all blocks (S, C) is at most $|\Pi(G)|n$, the inner loop of dynamic programming on lines 20 to 27 iterates $O(|\Pi(G)|n)$ times. As the total size of components C_i with $(S_i, C_i) \in \mathcal{A}(\Omega)$ is $O(n)$, the inner loop can be implemented to run in $O(n)$ time. The construction phase runs in $O(\min(n, |\Pi(G)|)n^2)$ time. □

For cost functions of clique-max-type and clique-sum-type, we can precompute the value $f(\Omega)$ of the clique function f for all PMCs Ω. Hence the total time complexity for those types is $O(|\Pi(G)|nm + |\Pi(G)|g_t)$, where g_t is the time complexity for computing $f(\Omega)$. For fill-in-type, the total time complexity is $O(|\Pi(G)|n^3)$, assuming that $f(e)$ can be computed in constant time, which is the case in all applications that we are aware of.

We present one additional remark in order to optimize the BT algorithm in practice

Proposition 20 ([42, 44])**.** *The potential maximal cliques in the input of BT-DP may be a subset $\Pi' \subseteq \Pi(G)$ of the actual potential maximal cliques of the input graph G. In this case, BT-DP considers exactly those minimal triangulations whose maximal cliques are in Π' and will return ∞ if no such minimal triangulation exists. The running time of BT-DP in this case is also $O(|\Pi'|n(m + g_t))$.*

When computing an optimal triangulation with respect to a cost function of clique-max-type or clique-sum-type, Proposition 20 allows to discard all potential maximal cliques Ω with $f(\Omega) > ub$, where ub is a known upper bound for the cost of an optimal triangulation.

4.4 Adaptation to Maximum Compatibility

In Section 3.5 we defined the maximum compatibility problem of phylogenetic characters and formulated the restricted binary characters version of the problem as finding an optimal triangulation with respect to a fill-in-type cost function. In [72] we showed that the general maximum compatibility problem remains NP-hard even in the case in which the partition intersection graph has a linear number of minimal separators and potential maximal cliques. Therefore, assuming P $\neq$ NP, the maximum compatibility problem cannot be solved with the BT algorithm without superpolynomial overhead. In this section we present a hybrid algorithm for the general maximum compatibility problem of phylogenetic characters, making use of Theorem 4 and maximum satisfiability (MaxSAT).

MaxSAT solvers are typically used for solving problems via MaxSAT encodings. A problem can be solved with a MaxSAT solver by first encoding it as MaxSAT, i.e., reducing it to an instance of the MaxSAT problem, and then using the solver to find an optimal solution to the MaxSAT instance [78]. In order to solve the maximum compatibility problem, we replace the BT-DP phase of the BT algorithm with a MaxSAT encoding of the characterization of minimal triangulations of Theorem 4. The encoding allows to find an optimal triangulation with a "global" point of view, avoiding the issue of double counting bad fill-edges corresponding to the same character.

Definition 20. ***(MaxSAT)*** *[78] A literal is a Boolean variable or a negation of a Boolean variable. A clause is a disjunction of literals. A clause is satisfied if at least one of its literals is true. A MaxSAT instance is a pair (F_S, F_H), where F_S is a set of soft clauses and F_H is a set of hard clauses. A solution of a MaxSAT instance is an assignment of the Boolean variables so that all hard clauses F_H are satisfied. An optimal solution is a solution that satisfies the greatest number of clauses in F_S over all solutions.*

We model maximum compatibility in MaxSAT so that each character $\mathcal{X}_i \in \mathfrak{C}$ has a corresponding Boolean variable Y_i with the interpretation that assigning $Y_i = 1$ corresponds to including $\mathcal{X}_i$ in the subset of compatible characters. The set of soft clauses is $\{(Y_i) \mid \mathcal{X}_i \in \mathfrak{C}\}$, and thus optimizing a solution of the encoding maximizes the number of compatible characters. Next we describe the hard clauses (and additional variables) that ensure that the subset of characters selected by Y_i is actually a compatible subset.

The hard clauses encode that there exists a minimal triangulation of $\text{int}(\mathfrak{C})$ that has no fill-edge of form $\{(\mathcal{X}_i, A), (\mathcal{X}_i, B)\}$ for any $\mathcal{X}_i$ with $Y_i = 1$. For the encoding, we assume that we have computed the set of potential maximal cliques $\Pi(\text{int}(\mathfrak{C}))$ with PMC-ENUM and the blocks of $\text{int}(\mathfrak{C})$ and relations $\mathcal{A}(\Omega)$ and $\Pi(S, C)$ with BT-DP. For each PMC $\Omega \in \Pi(\text{int}(\mathfrak{C}))$ we introduce an additional Boolean variable P_Ω with the interpretation that P_Ω is true if Ω is selected as a maximal clique of the triangulation. We ensure that the selected PMCs do not make the selected characters non-compatible with hard clauses

$$(\neg P_\Omega \vee \neg Y_i) \quad \text{for all } \Omega \in \Pi(\text{int}(\mathfrak{C})), \mathcal{X}_i \in \mathfrak{C} \text{ with } \{(\mathcal{X}_i, A), (\mathcal{X}_i, B)\} \in \Omega^2.$$

What remains is to encode that the selected PMCs actually form a minimal triangulation

of int($\mathfrak{C}$). For this, we make use of Theorem 4. We introduce an additional variable $B_{S,C}$ for each block (S, C) of int($\mathfrak{C}$) with the interpretation that $B_{S,C}$ is true if (S, C) is used in the recursion forming the minimal triangulation. We also introduce an additional variable $B_{S,C,\Omega}$ for all blocks and PMCs with $\Omega \in \Pi(S, C)$ with the interpretation that $B_{S,C,\Omega}$ is true if the minimal triangulation of (S, C) is achieved by recursing from Ω. We ensure that a PMC is chosen for each block by creating the hard clause

$$\left(\neg B_{S,C} \vee \bigvee_{\Omega \in \Pi(S,C)} B_{S,C,\Omega}\right) \quad \text{for all blocks } (S, C) \text{ of int}(\mathfrak{C}).$$

The variables $B_{S,C,\Omega}$ are propagated to variables P_Ω with the hard clauses

$$(\neg B_{S,C,\Omega} \vee P_\Omega).$$

The actual recursion to smaller blocks is enforced with the hard clauses

$$(\neg B_{S,C,\Omega} \vee B_{S_i,C_i}) \quad \text{for all } (S_i, C_i) \in \mathcal{A}_{R(S,C)}(\Omega).$$

Finally, the special block $(\emptyset, V(\text{int}(\mathfrak{C})))$ denoting the root case is forced to be used in the recursion with a hard clause $(B_{\emptyset,V(\text{int}(\mathfrak{C}))})$.

If follows from Proposition 13 and Theorem 4 that this encoding has a solution satisfying $|\mathfrak{C}'|$ soft clauses if and only if there exists a compatible subset $\mathfrak{C}' \subseteq \mathfrak{C}$ of the characters $\mathfrak{C}$. A maximum size compatible subset of characters can be constructed directly from the assignment of the Y_i variables of an optimal solution of the encoding. A corresponding triangulation can be constructed in polynomial time in a similar manner as in the construction phase of Algorithm 3.

Remark 3. *The encoding is a special case of MaxSAT known as Horn-MaxSAT [81]. While Horn-MaxSAT remains NP-hard, this fact indicates that the encoding is efficient in the sense that after guessing which soft clauses will be satisfied, the rest of the encoding can be solved in linear time [81].*

5 Implementation

In this chapter we describe our implementation of the BT algorithm, called *Triangulator* and available as open source under the MIT license in a GitHub repository [69]. The version of Triangulator that we introduce in this thesis is the fourth iteration of the implementation. It is significantly updated compared to the earlier versions submitted to PACE 2017 [33] and reported in [70, 71, 72]. The implementation consists of implementations of multiple preprocessing techniques for the problems, an optimized implementation of PMC-ENUM, and a relatively straightforward implementation of BT-DP.

We start by giving an overview of the implementation in order to give understanding on the organization of its various subroutines. In particular, the goal of the overview is to aid understanding of the experimental analysis of Triangulator in Chapter 7. After the overview, we describe the preprocessing techniques used in Triangulator, the optimizations used in PMC-ENUM, and other implementation choices that may have a significant impact on the performance of Triangulator.

5.1 Overview

We describe Triangulator in pseudocode as Algorithm 4. The goal of this description is to give a general problem-independent view of Triangulator that is sufficiently accurate on all of the problems. In particular, there is a notable difference between cost functions of clique-max-type and cost functions of other types because in clique-max-type, global lower bounds on the cost can be effectively made use of in a local context. Triangulator works in two phases, the preprocessing phase and the solving phase.

The preprocessing phase. Triangulator applies preprocessing techniques to make the input graph smaller and to compute lower bounds. The preprocessing loop on lines 7 to 22 considers graphs G_i from the preprocessing queue Q, which initially contains the input graph. First on lines 9 and 10 Triangulator computes a lower bound for the cost of a minimal triangulation of G_i with problem-specific lower bound algorithms. Then Triangulator computes a minimal triangulation H_i of G_i with the MCS-M algorithm (detailed in the next section) on line 11. The minimal triangulation H_i is used on line 12 to discard G_i if H_i is already an optimal triangulation, witnessed by the known lower bounds or by the chordality of G_i. Otherwise H_i is used to compute the atoms of the graph G_i on line 15. Atoms are components of the graph for which minimal triangulations can be considered independently of each other (detailed in the next section). If the graph G_i can be broken into multiple atoms, each is re-inserted into the preprocessing queue. Finally, Triangulator attempts to reduce the graph G_i by applying problem-specific preprocessing techniques.

Algorithm 4: Triangulator

Input : A graph G with possibly additional problem-specific data.
Output: An optimal triangulation of G.

Preprocessing Phase:
Let lb be lower bound for the cost of a triangulation, initialized to 0.
Let Q be a queue of graphs to be preprocessed.
Let S be a list of preprocessed graphs.
Let H_{ans} be a graph to store the optimal triangulation, initialized to G.
Q.push(G)
while Q *not empty* **do**
    $G_i \leftarrow Q$.pop()
    $loc\_lb \leftarrow$ ComputeLB(G_i)
    $lb \leftarrow \max(lb, loc\_lb)$
    $H_i \leftarrow$ MCS-M(G_i)
    if $H_i = G_i$ *or* $C(H_i) \leq loc\_lb$ *or (clique-max-type and* $C(H_i) \leq lb$*)* **then**
        $E(H_{ans}) \leftarrow E(H_{ans}) \cup E(H_i)$
        continue
    $A \leftarrow$ Atoms(G_i, H_i)
    if $|A| \geq 2$ **then**
        Q.push($A_1, \ldots, A_{|A|}$)
        continue
    if $R_i \leftarrow$ *Problem-specific-preprocessing*(G_i) **then**
        Q.push(R_i)
        continue
    S.insert(G_i)
Solving Phase:
Sort S in non-decreasing order of the number of vertices.
for $G_i \in S$ **do**
    $H_i \leftarrow$ LB-Triang(G_i)
    $ub_i \leftarrow C(H_i)$
    if *clique-max-type and* $ub_i \leq lb$ **then**
        $E(H_{ans}) \leftarrow E(H_{ans}) \cup E(H_i)$
        continue
    $\Delta \leftarrow$ MS-ENUM(G_i)
    $\Pi' \leftarrow$ PMC-ENUM(G_i, Δ, ub_i)
    $H_b, ans \leftarrow$ BT-DP(G_i, Π')
    if $ans < ub_i$ **then**
        $E(H_{ans}) \leftarrow E(H_{ans}) \cup E(H_b)$
        $lb \leftarrow \max(lb, ans)$
    else
        $E(H_{ans}) \leftarrow E(H_{ans}) \cup E(H_i)$
        $lb \leftarrow \max(lb, ub_i)$
return H_{ans}, lb

The solving phase. The solving loop on lines 25 to 39 considers the preprocessed graphs G_i in non-decreasing order of the number of vertices, in order to compute as good lower bounds as possible before solving the largest components. The solving loop starts by computing a minimal triangulation H_i of G_i with the LB-Triang algorithm (detailed in the next section) on line 26 to obtain an upper bound ub_i on the cost of the triangulation. This upper bound is used on line 28 to skip G_i if H_i is already optimal and in the PMC-ENUM algorithm to ignore the potential maximal cliques that cannot be used to obtain a triangulation with lower cost than ub_i. Lines 31 to 33 implement the actual BT algorithm as presented in Chapter 4, first enumerating all minimal separators, then enumerating the potential maximal cliques whose costs are less than ub_i, and finally computing an optimal triangulation with BT-DP. Note that if H_i is already optimal, BT-DP might return ∞, which is handled on lines 38 and 39.

5.2 Preprocessing

In this section we describe the preprocessing steps used by Triangulator in more detail. We first describe the preprocessing techniques that are applicable to all problems formulated as finding optimal minimal triangulations and then the problem-specific preprocessing techniques.

5.2.1 General Techniques

We start by describing the MCS-M and LB-Triang algorithms for computing minimal triangulations and the process of breaking a graph into atoms. Triangulator uses these techniques for preprocessing on all of the problems considered in this thesis. The reason for using two different algorithms for computing minimal triangulations is that in our preliminary experiments, we observed MCS-M to be faster than LB-Triang, but LB-Triang often produced triangulations with lower cost. Therefore we use LB-Triang in the solving phase to obtain good upper bounds immediately before applying the BT algorithm and MCS-M in the preprocessing phase where the graph still could potentially be decomposed to smaller atoms.

MCS-M is an algorithm that computes a minimal triangulation of a graph in $O(nm)$ time [11]. It works by finding a reverse elimination ordering of the graph by successively labeling its vertices with integers. In the beginning of its execution, all vertices have label 0. In each iteration, MSC-M chooses a vertex v with a greatest label to be the last one in the elimination ordering, and increments the labels of vertices u that can be reached from v via vertices that have not yet been added to the ordering and have smaller labels than v. This produces an elimination order from which a minimal triangulation can be constructed.

LB-Triang is also an algorithm that computes a minimal triangulation of a graph in $O(nm)$ time [13]. It works by considering vertices $v \in V(G)$ in any order and filling the

sets $\{N(C) \mid C \in \mathcal{C}(G \setminus N[v])\}$ into cliques [13]. We augment the LB-Triang algorithm with the *fill-in heuristic*. With the fill-in heuristic, in each step the vertex v that minimizes the number of added edges is chosen. In preliminary experiments we compared LB-Triang with the fill-in heuristic to LB-Triang with the minimum-degree heuristic [19] and to MCS-M. Of these three approaches, LB-Triang with the fill-in heuristic often produced triangulations with the smallest cost. The fill-in heuristic has been observed to be effective with triangulation minimalization as a post-processing step [19], but to the best of our knowledge it has not been applied together with LB-Triang before. The addition of the fill-in heuristic increases the time complexity of LB-Triang to $O(n^3)$.

The atoms of a graph generalize the concept of connected components in the context of minimal triangulations in the sense that no minimal triangulation has a fill-edge between two different atoms [100]. The atoms of a graph are computed by successively decomposing the graph by its *clique minimal separators*.

Proposition 21 ([100])**.** *Let G be a graph and $S \subseteq V(G)$ a vertex set that is both a minimal separator and a clique in G. The graph H is a minimal triangulation of G if and only if $V(H) = V(G)$ and*

$$E(H) = \bigcup_{C_i \in \mathcal{C}(G \setminus S)} E(H_i),$$

where H_i is a minimal triangulation of the induced subgraph $G[N(C_i) \cup C_i]$. Furthermore, $MC(H_i) \cap MC(H_j) = \emptyset$ for $i \neq j$ and $MC(H) = \cup_{C_i \in \mathcal{C}(G \setminus S)} MC(H_i)$.

A vertex set S as in Proposition 21 is called a clique minimal separator. By Proposition 21, if a graph contains a clique minimal separator S, all of its minimal triangulations can be formed by combining minimal triangulations of induced subgraphs $G[N(C_i) \cup C_i]$, where $C_i \in \mathcal{C}(G \setminus S)$ [100]. An atom of a graph is an induced subgraph resulting from successively decomposing the graph by its clique minimal separators. The set of atoms can be computed in $O(nm)$ time given a minimal triangulation of the graph [100]. Note that computation of the cost functions that we consider also decomposes in the manner of Proposition 21. The process of decomposing a graph into atoms generalizes multiple other techniques that could be considered independently, including decomposing a graph into connected components, eliminating vertices whose neighborhood is a clique (simplicial vertices), and decomposing a graph into biconnected components.

5.2.2 Treewidth

Next we describe the treewidth-specific preprocessing techniques used in Triangulator. The techniques for treewidth are based on computing lower bounds based on so-called degeneracy of the graph and on decomposing the graph with almost clique minimal separators.

The *degeneracy* of a graph G is the least number k such that all induced subgraphs of G have a vertex with at most k neighbors, and it can be computed in linear time [82]. The treewidth of G is at least the degeneracy of G [15]. We note that degeneracy generalizes

other lower bounds that could be considered independently, for example, the size of a maximum clique and the minimum degree [15].

On computing treewidth, a graph can be decomposed by *almost clique minimal separators*.

Proposition 22 ([18]). *If S is a minimal separator of a graph G and $S \setminus \{v\}$ is a clique for some vertex $v \in S$, then the treewidth of G is equal to the treewidth of the graph G' with $V(G') = V(G)$ and $E(G') = E(G) \cup S^2$.*

A vertex set S as in Proposition 22 is called an almost clique minimal separator. By Proposition 22, an almost clique minimal separator can be filled into a clique without affecting the treewidth of the graph. After filling an almost clique minimal separator into a clique, it becomes a clique minimal separator and thus allows decomposing the graph into atoms by Proposition 21. Note that all minimal separators of size two are almost clique minimal separators.

In Triangulator, we first employ an algorithm that searches for almost clique minimal separators contained in the neighborhoods of vertices. Such separators can be found in the neighborhood of a vertex v by considering the sets $\{N(C) \mid C \in \mathcal{C}(G \setminus N[v])\}$. Then we use an $O(n^2m)$ algorithm [18] to find all almost clique minimal separators. The algorithm guesses the non-clique vertex, removes it from the graph, and finds clique separators of the resulting graph in $O(nm)$ time.

We note that applying these rules in conjunction with decomposing the graph into atoms results in each atom being a clique or having a minimum degree of at least three. Therefore, the preprocessing phase of Triangulator is able to find and prove the optimality of an optimal tree decomposition of width ≤ 2 in polynomial time if such a tree decomposition exists.

5.2.3 Minimum Fill-In

Triangulator uses two preprocessing techniques that are specific to minimum fill-in. First, we note that if a graph is not chordal, the minimum fill-in is at least one. Second, we use a reduction from [17] to remove *near clique minimal separators* whose definition is similar but not the same as that of almost clique minimal separators.

Proposition 23 ([17]). *If S is a minimal separator of a graph G such that $S \subseteq N(v)$ for a vertex $v \in V(G)$ and $|S^2 \setminus E(G)| = 1$, then the minimum fill-in of G is equal to the minimum fill-in of the graph G' with $V(G') = V(G)$ and $E(G') = E(G) \cup S^2$.*

A vertex set S as in Proposition 23 is called a near clique minimal separator. Near clique minimal separators are used in the same manner as almost clique minimal separators, allowing the decomposition of the graph into atoms by Proposition 21 after filling them into cliques. Triangulator finds near clique minimal separators of a graph G by considering the sets $\{N(C) \mid C \in \mathcal{C}(G \setminus N[v])\}$ for each vertex $v \in V(G)$.

We note that also in the case of minimum fill-in, applying the reduction of Proposition 23 in conjunction with decomposing a graph into atoms will result in all atoms being cliques or having minimum degree of at least 3. Also, it is guaranteed that the preprocessing phase of Triangulator is able to find and prove the optimality of an optimal triangulation if it has at most one fill-edge. This is due to the fact that if the graph has no simplicial vertices, then the single fill-edge must be contained in a near clique minimal separator.

5.2.4 Phylogenetic Character Compatibility

On the perfect phylogeny and maximum compatibility problems we simplify the character-state matrix before computing the PI-graph. We remove the character states that correspond to only one taxon and the unary characters. Removing character states corresponding to only one taxon is correct due to the equivalence of missing data and states that correspond to only one taxon [96], and removing unary characters is the so-called trivial character technique from [97].

On perfect phylogeny, we compute the degeneracy of the PI-graph. If the degeneracy is at least $|\mathfrak{C}|$, then the treewidth of the PI-graph is at least $|\mathfrak{C}|$ [15], and therefore any triangulation of the PI-graph has a clique with at least $|\mathfrak{C}| + 1$ vertices. In this case, the characters do not admit perfect phylogeny because a clique with $|\mathfrak{C}|+1$ vertices necessarily contains two vertices corresponding to the same character.

We remark that even in the case of multi-state maximum compatibility a graph whose heuristically computed triangulation does not add any fill-edges between vertices corresponding to a same character can be removed on line 12 of Algorithm 4.

5.2.5 Other Problems

We use two simple observations for lower bounds on generalized hypertreewidth and treelength.

Generalized hypertreewidth lower bounds are obtained using the fact that if a primal graph of a hypergraph is not chordal, then the hypergraph has a generalized hypertreewidth at least 2 [41].

Treelength lower bounds are obtained using the fact that if a graph is not chordal, then its treelength is at least 2 [34].

5.3 Optimizing PMC-ENUM

The implementations of MS-ENUM and BT-DP in Triangulator are very similar to their representations as pseudocode in Algorithms 1 and 3, respectively. On the other hand, for PMC-ENUM we use non-trivial techniques to speed up the implementation in practice.

In particular, Triangulator always enumerates all minimal separators in MS-ENUM but aims to restrict the set of enumerated PMCs in PMC-ENUM.

By Proposition 20, BT-DP does not need to be given all PMCs in the input, as long as all PMCs that might be maximal cliques of an optimal triangulation are included in the input. Due to this we will not enumerate the PMCs that cannot be used to improve the current known upper bound. In clique-max-type and clique-sum-type this means that if we already have a triangulation with a cost ub, then we will not enumerate any PMCs Ω with $f(\Omega) \geq ub$. For fill-in-type cost functions we can use the same filtering with a slight extension of the notation by letting $f(\Omega) = \sum_{e \in \Omega^2 \setminus E(G)} f(e)$ for a vertex set Ω.

Recall that in PMC-ENUM we gradually remove vertices $a \in V(G_i)$ from the graph G_i to use induction over induced subgraphs $G_1, \ldots, G_n$. For the order of removing vertices we use an ordering computed by the MCS algorithm [11]. This ensures the required property that $G_i \setminus \{a\}$ is connected and an interesting property, which we will make use of, that a is a so-called *OCF-vertex* of G_i, or equivalently that $N[a]$ is a PMC of G_i [11].

The bottleneck of PMC-ENUM is the COMBINE(Δ_S, Δ_T) subroutine corresponding to case 3 of Corollary 4, where PMCs of G_i are formed by unions of pairs $S \in \Delta_S$, $T \in \Delta_T$, where $S \in \Delta(G_i)$, $a \notin S$, $S \notin \Delta(G_i \setminus \{a\})$, $(T \cup \{a\}) \in \Delta(G_i)$, and $a \notin T$. We refer to the elements of Δ_S as S-separators and to the elements of Δ_T as T-separators. Note that T-separators are not necessarily minimal separators of G_i, but they are minimal separators of $G_i \setminus \{a\}$. By the proof of Theorem 2, each S-separator has two full components C_a and C_Ω, and in the COMBINE subroutine we are interested in finding PMCs $\Omega = S \cup T$, where S is a S-separator, T is a T-separator, and $T \subseteq S \cup C_\Omega$.

Let $C_\Omega(S)$ denote the full component of an S-separator S that does not contain a. To speed up COMBINE, we arrange the S-separators into a directed acyclic graph (DAG) so that if S' is a descendant of S in the DAG, then $C_\Omega(S') \subset C_\Omega(S)$. In particular, for an S-separator S, we let the minimal separators $\{N(C) \mid C \in \mathcal{C}(G_i \setminus (S \cup N(v))), C \subset C_\Omega(S) \mid v \in S\}$ that are also S-separators be its children in the DAG. Note that this is a slightly modified version of the rule to generate minimal separators in MS-ENUM. In particular, it only generates minimal a,b-separators for all $b \in C_\Omega(S)$ [12]. Next we show that this DAG has an unique root $N(a)$.

Lemma 9. *Let a be an OCF-vertex of a graph G. If there is a minimal separator $S \in \Delta(G) \setminus \Delta(G \setminus \{a\})$ with $a \notin S$, then $N(a) \in \Delta(G) \setminus \Delta(G \setminus \{a\})$ and $S \not\subset N(a)$.*

Proof. Let $b \in V(G)$ be a vertex such that $S \in \Delta(G) \setminus \Delta(G \setminus \{a\})$ is a minimal a,b-separator. To prove that $N(a)$ is a minimal a,b-separator, suppose the contrary that there is a vertex $v \in N(a)$ such that there is no path from v to b outside of $N[a]$. Clearly $v \notin S$, so v is in the same full component C_a of S as a. Because a is an OCF-vertex, there is a path from v to every vertex of $N(a)$ outside of $N[a]$. If all intermediate vertices of such paths are in C_a, then S would have a full component containing $N(v) \setminus S$ in $G \setminus \{a\}$, which would imply $S \in \Delta(G \setminus \{a\})$. Thus there is a path from v to a vertex in S outside of $N[a]$, and therefore there is a path from v to b outside of $N[a]$, which is a contradiction. Therefore $N(a)$ is a minimal a,b-separator. Since also S is a minimal a,b-separator, it cannot be a

subset of $N(a)$. To prove that $N(a) \notin \Delta(G \setminus \{a\})$, let C_b be the full component of $N(a)$ containing b. Note that $S \subseteq N(a) \cup C_b$, so if there would be another full component C_o of $N(a)$ in $G \setminus \{a\}$, then it could be extended into a full component of S, distinct of the full component of S containing b, and therefore $S \in \Delta(G \setminus \{a\})$ would hold. □

By Lemma 9, the minimal separator $N(a)$ is the unique root of the DAG, i.e., all other S-separators in the DAG are its descendants. The connectedness of the DAG follows from the fact that for an S-separator S, all minimal a,b-separators S' for arbitrary b that have a full component C with $C_\Omega(S) \subset C$ and $a \notin C$ are also S-separators. Note that the non-triviality of this DAG is that it has a unique root and each S-separator in it has at most n^2 direct descendants (and in practice much less).

Next we find a spanning tree of the DAG, excluding the S-separators with $f(S) \geq ub$. Then we have constructed in $O(|\Delta_S|n^3)$ time a tree of S-separators, where the full components C_Ω strictly decrease by inclusion on paths from the root to the leaves. To find the PMCs $\Omega = S \cup T$, we traverse this tree for each $T \in \Delta_T$ with $f(T) < ub$. We use the inclusion property of C_Ω to return early in this traversal. We also use it to speed up the checking of the cliquish condition by using the observation that the connectivity provided by $G_i[C_\Omega \setminus T]$ decreases when going down the tree. We remark that even though we do not provide extensive proofs of these techniques, we have tested the equivalence of the optimized Combine with a naive Combine with millions of random graphs with 20-30 vertices.

5.4 Computing Edge Covers

Generalized hypertreewidth and fractional hypertreewidth have non-trivial clique functions. Computing the minimum edge cover in generalized hypertreewidth corresponds to the NP-hard set cover problem [86]. Computing the minimum fractional edge cover in fractional hypertreewidth corresponds to the fractional relaxation of the set cover problem, which can be solved in polynomial time by linear programming [86].

In Triangulator, we use the CPLEX solver [61] to compute minimum edge covers on generalized hypertreewidth and minimum fractional edge covers on fractional hypertreewidth. The (fractional) edge covers are computed for all PMCs between the PMC-ENUM and the BT-DP phases. We aim to reduce the overhead of CPLEX by reusing the instance object with the same variables (corresponding to edges), only changing the constraints (corresponding to the vertices of the PMC) between each instance. We also provide known lower and upper bounds to CPLEX.

As mentioned in the previous section, PMC-ENUM uses the clique function to discard the PMCs whose cost is at least the known upper bound. Due to the computational hardness of computing edge covers, we do not do this when computing fractional hypertreewidth, and for generalized hypertreewidth we instead use a lower bound given by the size of a maximal independent set contained in the PMC.

5.5 Low-level Implementation Details

Triangulator consists of more than 5000 lines of C++11 code. The only external libraries that Triangulator uses are CPLEX 12.7.1 [61] for computing (fractional) edge covers and MaxHS 3.0 [28][1] for MaxSAT solving for multi-state maximum compatibility.

Triangulator makes an extensive use of a handwritten bitset class for representing sets of vertices in a graph. Graphs are represented with adjacency matrices that are bitsets, and many graph search subroutines are implemented in an $O(n + n^2/w)$-time manner, where $w = 64$ is the word size. We believe that this implementation can be more efficient than adjacency list based implementations in graphs with less than ≈ 256 vertices. For large sparse graphs, an adjacency list based implementation is likely more efficient.

For basic data structures such as vector, queue, and map, Triangulator makes use of the C++ standard library. In general, the use of binary search tree based data structures is avoided in performance intensive parts, replacing them with sorting and binary search as far as possible. In the MS-ENUM algorithm, a custom open addressing hashtable is implemented for storing the set of enumerated minimal separators.

[1] From https://github.com/fbacchus/MaxHS.

6 Empirical Comparison

We empirically compare Triangulator to other exact implementations for solving treewidth, minimum fill-in, generalized hypertreewidth, fractional hypertreewidth, total table size of Bayesian networks, perfect phylogeny, and maximum compatibility of phylogenetic characters. We also experiment on computing treelengths of graphs, although to the best of our knowledge no other implementations of exact algorithms for treelength are available. At the time of writing, complete data of all experiments is available in the GitHub repository [69], allowing for reconstructing all tables and plots reported here. Appendix A contains a table summarizing the numbers of solved instances, timeouts, and memouts/runtime errors for all implementations tested in the experiments.

We first describe the experimental setup and the used datasets, and then present the results of experiments for each problem. In this chapter we focus on comparing the overall performance of Triangulator to other implementations, and in Chapter 7 we specifically analyze Triangulator.

6.1 Empirical Setup

All experiments were ran single-threaded on computers with 2.4-GHz Intel Xeon E5-2680-v4 processors, with a 1-hour per-instance time limit and 64-GB memory limit. The implementations were compiled using GCC 7.3.0 with optimization flags `-O2` and `-march=native`. For all problems and all solved instances, the tested implementations agreed on the cost of an optimal solution.

6.2 Benchmarks

Next we describe the datasets that we use in the evaluation of the implementations.

PACE 2017 [33] was an algorithm implementation competition that included tracks for exact computing of treewidth and minimum fill-in. We use the 200 instances of the exact treewidth track[2] to evaluate performance on treewidth and the 100 instances of the exact minimum fill-in track[3] to evaluate performance on minimum fill-in. We also use 100 "bonus" treewidth instances[4] published by the PACE 2017 organizers after the competition to evaluate performance on treewidth. All of the PACE 2017 instances are either graphs directly from real-world applications or subgraphs of them [33].

[2]From https://github.com/PACE-challenge/Treewidth-PACE-2017-instances.
[3]From https://pacechallenge.org/2017/minimum-fill-in/.
[4]From https://github.com/PACE-challenge/Treewidth-PACE-2017-bonus-instances.

A dataset of 150 "**named graphs**"[1] has been used for example in PACE 2016 [32]. The named graphs have been extracted from the Sage graph package and they include various graphs of general interest, e.g., constructions from the graph theory literature and the adjacency graph of the countries of the world [32]. We use the named graphs to evaluate performance on treewidth, minimum fill-in, and treelength.

Tree decompositions of **control flow graphs** have applications in compilers [74, 75]. We use 1817 control flow graphs of C functions[2] to evaluate performance on treewidth. Some of these graphs were used in PACE 2016 [32].

The **DIMACS** graph coloring instances [64][3] are a well-known dataset in empirical evaluation of graph algorithms. We use the DIMACS graphs for evaluating performance on treewidth, minimum fill-in, and treelength.

We use 24 discrete Bayesian networks from the **BNlearn** repository [95][4] to evaluate performance on treewidth and total table size. These instances have sources in specific applications of Bayesian networks [95], and they are a well-known dataset in evaluating algorithms for Bayesian networks.

Hyperbench is a repository of 3070 hypergraphs for which computing generalized hypertreewidth and fractional hypertreewidth are well-motivated [40][5]. Out of the 3070 hypergraphs, 1035 arise from conjunctive queries in databases and 2035 from constraint satisfaction problem instances [40]. We use the hypergraphs to evaluate performance on generalized hypertreewidth and fractional hypertreewidth and their primal graphs to evaluate performance on treewidth.

For phylogenetic character compatibility, we use phylogenetic data generated by a well-known tool **ms** [60][6] and 5 real-world instances[7], used for example in [24, 85].

6.3 Treewidth

On computing treewidth, we compare Triangulator to Positive Instance Driven Dynamic programming for Treewidth (PIDDT) [33, 99][8], p17 [33][9] and QuickBB [48][10]. PIDDT is a "positive instance driven" implementation of the BT algorithm [33, 99]. Differentiating it from Triangulator, PIDDT does not list all minimal separators and potential maximal cliques. In particular, it lists only the blocks whose realization has treewidth at most the treewidth of the graph [99]. PIDDT achieves this by listing so-called O-blocks whose

[1] From https://github.com/freetdi/named-graphs.
[2] From https://github.com/freetdi/CFGs.
[3] From https://mat.tepper.cmu.edu/COLOR/instances.html.
[4] From https://www.bnlearn.com/bnrepository/.
[5] From http://hyperbench.dbai.tuwien.ac.at/.
[6] From http://home.uchicago.edu/~rhudson1/source/mksamples.html.
[7] From http://sat.inesc-id.pt/~miguel/phylo/.
[8] From https://github.com/TCS-Meiji/PACE2017-TrackA.
[9] From https://github.com/freetdi/p17.
[10] From http://www.hlt.utdallas.edu/~vgogate/quickbb.html.

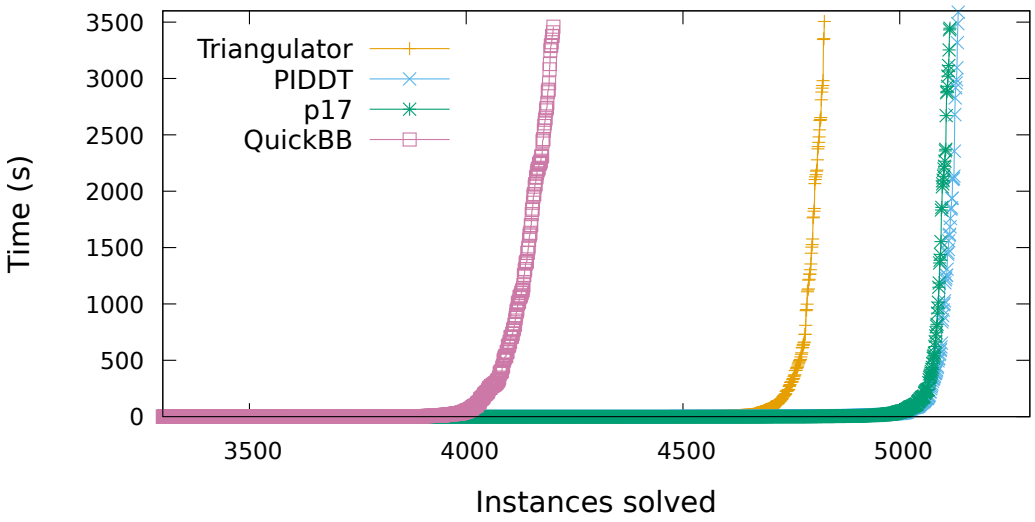

Figure 6.1: Comparison of the empirical performance of Triangulator, PIDDT, p17, and QuickBB in computing treewidth.

number is not known to be bounded by the number of minimal separators or potential maximal cliques [99]. PIDDT took the second place in PACE 2017 [33]. The p17 implementation took the first place in PACE 2017 [33]. It is an updated version of the first place implementation of PACE 2016 [32], which is an optimized implementation of the algorithm of Arnborg et al. that works in $O(n^{k+2})$ time, where k is the treewidth [4]. The p17 implementation has been analyzed theoretically and empirically in [6]. QuickBB is a branch-and-bound algorithm for computing treewidth published in 2004 [48]. QuickBB searches the space of the elimination orderings of the graph, with several pruning heuristics [48]. We note that in [71] we also compared to a MaxSAT approach for treewidth [9], which was clearly outperformed by Triangulator, which is why we do not include MaxSAT in our comparison.

We use control flow graphs, DIMACS graphs, primal graphs of hypergraphs from Hyperbench, named graphs, moral graphs of Bayesian networks from BNlearn, PACE 2017 treewidth instances, and PACE 2017 bonus treewidth instances to compare the performance of the implementations on computing treewidth. Figure 6.1 shows an overview of the performance of the implementations on the instances. The figure shows for each of the implementations number of instances solved (x-axis) within a time limit (y-axis). Table 6.1 reports the dataset specific numbers of solved instances by the implementations, including also VBS denoting the virtual best solver, i.e., the number of instances solved by at least one of the implementations. PIDDT and p17 solve the most instances, with Triangulator 290 instances behind p17 and QuickBB 625 instances behind Triangulator. The largest difference between Triangulator and the top implementations is in instances from PACE 2017. The reason for this may be that the PACE 2017 instances were carefully selected to be solvable by the best solvers but still very hard [33].

Triangulator solves 6 instances that PIDDT does not solve, 5 of them in the DIMACS

Table 6.1: Number of solved instances for each dataset on treewidth.

Instance set	#Instances	Approach				
		Triangulator	PIDDT	p17	QuickBB	VBS
CFG	1817	1817	1817	1817	1816	1817
DIMACS	80	36	35	25	29	41
HyperBench	3070	2740	2886	2879	2236	2886
Named graphs	150	112	122	116	82	122
Moral graphs	24	21	24	24	16	24
PACE 2017	200	92	200	196	21	200
PACE 2017 bonus	100	8	51	59	1	63
Total	5441	4826	5135	5116	4201	5153

dataset and 1 in the PACE 2017 bonus dataset. Triangulator solves 20 instances that p17 does not solve, 12 of them in DIMACS, 1 in named graphs, 1 in Hyperbench, 3 in PACE 2017, and 3 in PACE 2017 bonus.

6.4 Minimum Fill-In

On computing minimum fill-in, we compare Triangulator to Minimum Chordal Completion Polytope (MCCP) [10][1] and to Positive Instance Driven Dynamic programming for Minimum fill-in (PIDDM) [33, 99][2]. MCCP is an integer programming approach for minimum fill-in published in 2019 [10]. MCCP is based on an exponential number of inequalities on the cycles of the graph, encoded in a lazy-constraint generation manner [10]. PIDDM is a "positive instance driven" implementation of the BT algorithm by the same authors as PIDDT for treewidth [33, 99]. PIDDM took the first place in the minimum fill-in track of PACE 2017 [33].

We use the PACE 2017 minimum fill-in instances, DIMACS instances and the named graphs to compare performance on minimum fill-in. The results of the experiments are summarized in Figure 6.2, comparing the overall scalability of the implementations and in Table 6.2 in which we report dataset-specific numbers of solved instances, including VBS

[1] Obtained directly from the authors.

[2] From https://github.com/TCS-Meiji/PACE2017-TrackB.

Table 6.2: Number of solved instances for each dataset on minimum fill-in.

Instance set	#Instances	Approach			
		Triangulator	MCCP	PIDDM	VBS
DIMACS	80	36	15	34	36
Named graphs	150	106	78	100	109
PACE 2017	100	61	29	61	63
Total	330	203	122	195	208

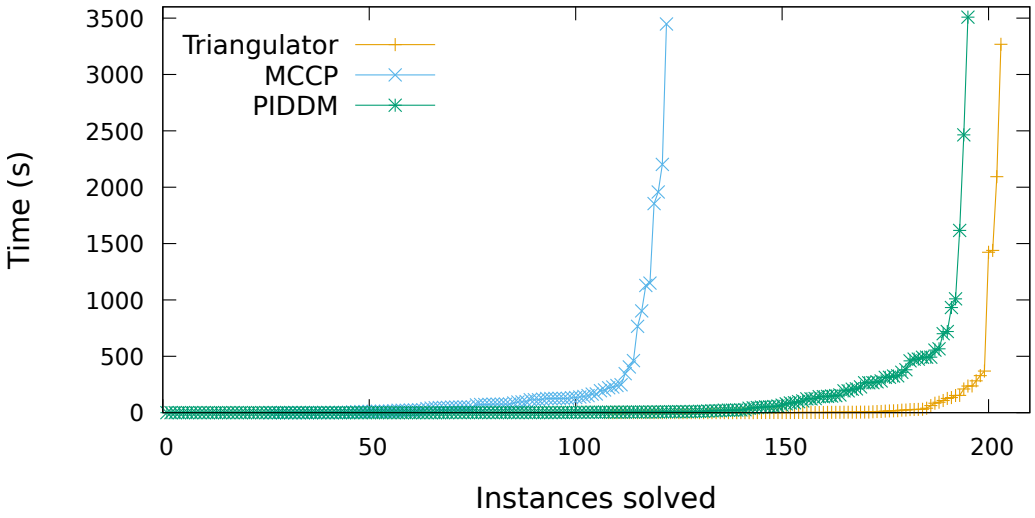

Figure 6.2: Comparison of the empirical performance of Triangulator, MCCP, and PIDDM on minimum fill-in.

denoting the virtual best solver, i.e., the number of instances solved by at least one of the implementations.

Triangulator solved the most instances overall, with PIDDM solving 8 instances less. Both Triangulator and PIDDM solved significantly more instances that MCCP. Triangulator slightly outperforms PIDDM on the DIMACS and named graphs datasets. On PACE 2017 instances, both of them solve 61 instances, each solving one instance that another did not solve. In PACE 2017, the winning implementation (PIDDM) solved 54 instances [33]. The "positive instance driven" paradigm is relatively less efficient in minimum fill-in than in treewidth. We suspect that the reason for this is that in minimum fill-in the cost of a triangulation is accumulated "more globally" than in treewidth, which is why a smaller portion of the blocks can be pruned via the positive instance driven approach.

MCCP solved two instances that neither Triangulator nor PIDDM solved, both in the Named graphs set. These graphs were the Kneser Graph $KG_{8,3}$ [68] and the Sims–Gewirtz Graph [47]. We suspect that BT-based approaches are relatively worse on these graphs because the graphs have a high number of minimal separators compared to the number of vertices: both of the graphs have 56 vertices and more than 16,000,000 minimal separators.

6.5 Generalized and Fractional Hypertreewidth

On generalized hypertreewidth (GHTW) and fractional hypertreewidth (FHTW), we evaluated the performance of Triangulator and multiple other algorithms on the Hyperbench instance set [40]. We first discuss this setup and the results in general and then specifically on GHTW and FHTW.

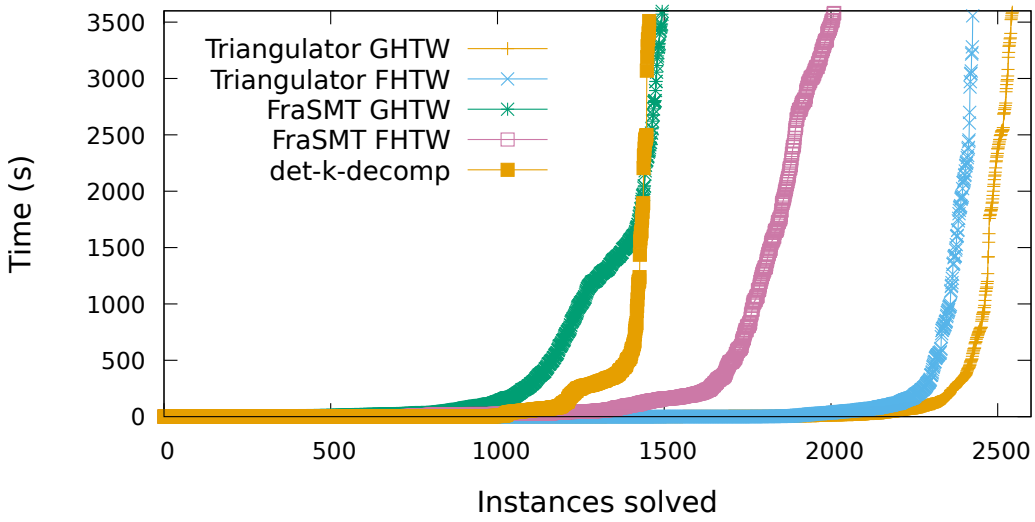

Figure 6.3: Comparison of empirical performance of Triangulator and FraSMT on generalized and fractional hypertreewidth and of det-k-decomp on hypertreewidth.

We compare to FraSMT [38][1], GlobalBIP, LocalBIP, BalSep [40][2], and det-k-decomp [51][3]. FraSMT is an elimination ordering based satisfiability modulo theory (SMT) [7] encoding for generalized and fractional hypertreewidth published in 2018 [38]. FraSMT uses Z3 [30] as the underlying SMT solver. FraSMT was developed originally for computing fractional hypertreewidth, but the same encoding supports also generalized hypertreewidth by enforcing integrality of the variables [38]. We note that the generalized hypertreewidth encoding of FraSMT is used as a basis for the implementation that took the first place in the PACE 2019 challenge for computing hypertreewidth [36, 94]. GlobalBIP and LocalBIP are algorithms for computing generalized hypertreewidth based on exploiting the so-called bounded intersection property (BIP) of hypergraphs, which is observed to be small in many instances of Hyperbench [40]. BalSep is a "balanced separator" based algorithm for generalized hypertreewidth [40]. GlobalBIP, LocalBIP, and BalSep were published in 2019 [40]. Det-k-decomp is a backtracking algorithm for computing hypertreewidth published in 2008 [51]. Det-k-decomp works in $O(n^{2k+2})$ time, where k is the hypertreewidth [51]. GlobalBIP, LocalBIP, BalSep, and det-k-decomp have only decision version implementations, i.e., they take the value k as a parameter and decide if the width is at most k. For the comparison, we run them starting with $k = 1$ and increasing k until the algorithm reports a positive answer.

Figure 6.3 summarizes the overall scalability of the implementations on the Hyperbench dataset, excluding GlobalBIP, LocalBIP, and BalSep because they solved less than 1050 instances. Triangulator solved 2544 instances on generalized hypertreewidth and 2425 instances on fractional hypertreewidth. All other approaches solved significantly less instances than Triangulator, FraSMT being the second solving 1494 instances on generalized

[1]From https://github.com/daajoe/frasmt.
[2]From https://github.com/TUfischl/newdetkdecomp.
[3]From https://github.com/TUfischl/newdetkdecomp.

hypertreewidth and 2010 instances on fractional hypertreewidth. Det-k-decomp solved 1455 instances on hypertreewidth. The inequalities $\mathtt{fhw}(\mathcal{G}) \leq \mathtt{ghw}(\mathcal{G}) \leq \mathtt{hw}(\mathcal{G})$ hold in all of the results.

6.5.1 Generalized Hypertreewidth

Table 6.3 reports the number of solved instances on generalized hypertreewidth grouped by the Hyperbench category of the instance, including VBS denoting the virtual best solver, i.e., the number of instances solved by at least one of the implementations. SQL-Share and SPARKQL are conjunctive query instances arising from applications in specific databases, CQ-other is conjunctive queries from other applications and CQ-rand are random conjunctive queries [40]. CSP-app are hypergraphs arising from CSPs representing real combinatorial optimization instances and CSP-rand are hypergraphs arising from random CSPs [40].

Triangulator solved the most instances in all of the categories. Triangulator matched the virtual best solver in all categories except CSP-app. The three instances that Triangulator did not solve but VBS solved were instances arising from the Kakuro game, all of which had 148 vertices, GHTW = 3, and were solved by LocalBIP. In these three instances, Triangulator was terminated due to time limit in MS-ENUM after enumerating more than 200,000,000 minimal separators.

Table 6.4 reports the number of solved instances grouped by the generalized hypertreewidth of the instance. Triangulator outperforms other implementations especially on instances with generalized hypertreewidth at least 6. The other implementations solved a total of nine instances with generalized hypertreewidth at least 6, while Triangulator solved 772 such instances.

Table 6.3: Number of solved instances on generalized hypertreewidth grouped by Hyperbench category.

Category	#Instances	Approach					
		Triangulator	FraSMT	BalSep	G-BIP	L-BIP	VBS
SQLShare	290	290	289	284	290	290	290
SPARKQL	70	70	70	70	70	70	70
CQ-other	175	175	175	103	175	175	175
CQ-rand	500	439	279	225	139	155	439
CSP-app	1090	732	388	129	113	124	735
CSP-other	82	35	20	18	23	21	35
CSP-rand	863	803	273	183	48	66	803
Total	3070	2544	1494	1012	858	901	2547

Table 6.4: Number of solved instances on generalized hypertreewidth grouped by generalized hypertreewidth.

GHTW	#Instances	Approach				
		Triangulator	FraSMT	BalSep	G-BIP	L-BIP
1	490	490	489	407	490	490
2	236	236	228	228	233	236
3	310	307	281	248	120	167
4	358	358	330	102	15	8
5	381	381	160	24	0	0
6	443	443	6	2	0	0
7	258	258	0	0	0	0
8	62	62	0	1	0	0
9	9	9	0	0	0	0
Unknown	523	0	0	0	0	0
Total	3070	2544	1494	1012	858	901

6.5.2 Fractional Hypertreewidth

Table 6.5 reports the number of solved instances on fractional hypertreewidth grouped by the Hyperbench category of the instance. Triangulator solved the most instances in all categories, solving 415 instances more than FraSMT. However, FraSMT was closer to Triangulator than on generalized hypertreewidth, solving 94 instances that Triangulator did not solve. Of these instances, 79 are in the CSP-rand category, have 130 vertices, FHTW between 7 and 9, and more than 15,000,000 minimal separators. In those instances, Triangulator was terminated due to time limit in PMC-ENUM.

Table 6.5: Number of solved instances on fractional hypertreewidth grouped by Hyperbench category.

Instance set	#Instances	Approach		
		Triangulator	FraSMT	VBS
SQLShare	290	290	290	290
SPARKQL	70	70	70	70
CQ-other	175	175	175	175
CQ-rand	500	461	302	461
CSP-rand	863	763	598	842
CSP-app	1090	637	550	650
CSP-other	82	29	25	31
Total	3070	2425	2010	2519

Table 6.6: The number of nodes, optimal total table size, and runtimes of the implementations on computing total table size of Bayesian networks.

Instance	n	TTS	Approach	
			Triangulator	EDFS
alarm	37	996	0.01	0.88
barley	48	17,140,796	0.12	2678.13
child	20	642	0.03	0.99
hailfinder	56	9,406	0.03	5.21
hepar2	70	2,617	0.05	0.36
insurance	27	23,880	0.02	1.93
mildew	35	3,400,464	0.14	7.04
pathfinder	109	182,641	0.06	4.85
sachs	11	216	0.02	0.83
water	32	3,028,305	0.05	4.60
win95pts	76	2,684	0.04	9.10

6.6 Total Table Size

On computing total table size of Bayesian networks, we compare Triangulator to extended depth first search (EDFS) algorithm [77][1]. EDFS is a branch-and-bound algorithm for total table size published in 2017 [77]. EDFS is based on a search over elimination orderings [77]. EDFS extends earlier work [88] by a new maximal clique maintenance algorithm and new pruning rules for the search [77].

We compare Triangulator to EDFS on the dataset of 24 discrete Bayesian networks from BNlearn. Table 6.6 reports the running times of the implementations on solved instances with at least 10 nodes. Both of the implementations solved 15 of the instances, though Triangulator solved all of them in less than a second and EDFS used over 2000 seconds on the "barley" instance.

We note that on computing treewidth, Triangulator solved 21 of the Bayesian network instances and PIDDT solved all 24 of them. To motivate the exact computation of total table size instead of treewidth, we remark that the tree decomposition resulting from optimizing for treewidth in the "barley" instance had total table size of 60,457,895, which is roughly 3.5 times the optimal total table size.

6.7 Phylogenetic Character Compatibility

We consider three variants of the phylogenetic character compatibility problem: perfect phylogeny, maximum compatibility of binary characters, and maximum compatibility of multi-state characters. We note that multi-state maximum compatibility, which is the

[1]From https://bitbucket.org/chaoli/optimaltriangulation/src/master/

only variant we considered in [72], is the most general of the three variants, subsuming perfect phylogeny and binary maximum compatibility.

We compare to "Minsep IP" triangulation–integer programming hybrid approach [53, 56], "Bin IP" integer programming approach [54, 97], and to PerftPhy [26][1]. Minsep IP is an integer programming encoding for the multi-state maximum compatibility problem [53, 56]. Minsep IP makes use of the PI-graph approach for character compatibility, but instead of using potential maximal cliques it uses a characterization of minimal triangulations that is based on maximal independent sets in a graph whose vertices correspond to the minimal separators of the PI-graph [53]. Bin IP is an integer programming encoding for the binary maximum compatibility problem [54]. The encoding operates on the character-state matrix level, making use of the so-called four-gamete condition [54]. The Bin IP encoding has been extended for multi-state characters, reducing the multi-state case to the binary case with additional constraints [97]. For both of the integer programming approaches CPLEX 12.7.1 [61] is used as the integer programming solver. We were unable to obtain the original implementations for Bin IP and Minsep IP so we re-implemented the algorithms ourselves.[2] We paid attention to implement all of the optimizations and preprocessing techniques of these algorithms as presented in [56, 97]. PerftPhy is an implementation of the algorithm of Kannan and Warnow [65] for perfect phylogeny [26]. We note that in [72], we also compared to a pseudo-Boolean optimization (PBO) approach for multi-state maximum compatibility [85]. We do not report the results for PBO here because the encoding did not scale to instances with more than 40 taxa [72].

For experiments on each of the three variants we use the data generator ms [60]. Used in the fashion of Gusfield et al. [53, 56, 97], the data generator has 5 parameters: n, m, k, r, and p. The parameters n, m, and k are the number of taxa, the number of characters, and the maximum arity of the characters. The recombination parameter r intuitively controls how close the data is to perfect phylogeny, i.e., the generated data admits perfect phylogeny if $r = 0$, and in general increasing r makes the probability of perfect phylogeny smaller and the size of a maximum compatible subset of characters smaller. The parameter p is the probability of an entry in the character-state matrix being missing, i.e., the expected number of missing entries is pnm. To reduce the parameter space, we will use $n = m$, following [53, 97], and in all cases except binary maximum compatibility we use $p = 0$.

6.7.1 Perfect Phylogeny

On perfect phylogeny, we compare the scalability of the implementations on parameters $n = m$, k, and r. We chose the values $n = m = 400$, $k = 20$, and $r = 0.25$ as the baseline and generated three instance sets with ms. In the first set, the values of $n = m$ vary from 40 to 800. In the second, the value of k varies from 2 to 40. In the third, the value of r varies from 0 to 0.5. For each considered combination of parameters, we generated 20 instances. These combinations of parameters turned out to be reasonable

[1] From https://csiflabs.cs.ucdavis.edu/~gusfield/perfectphy.tar.gz.
[2] The re-implementations are available at https://github.com/laakeri/phylogeny-aaai.

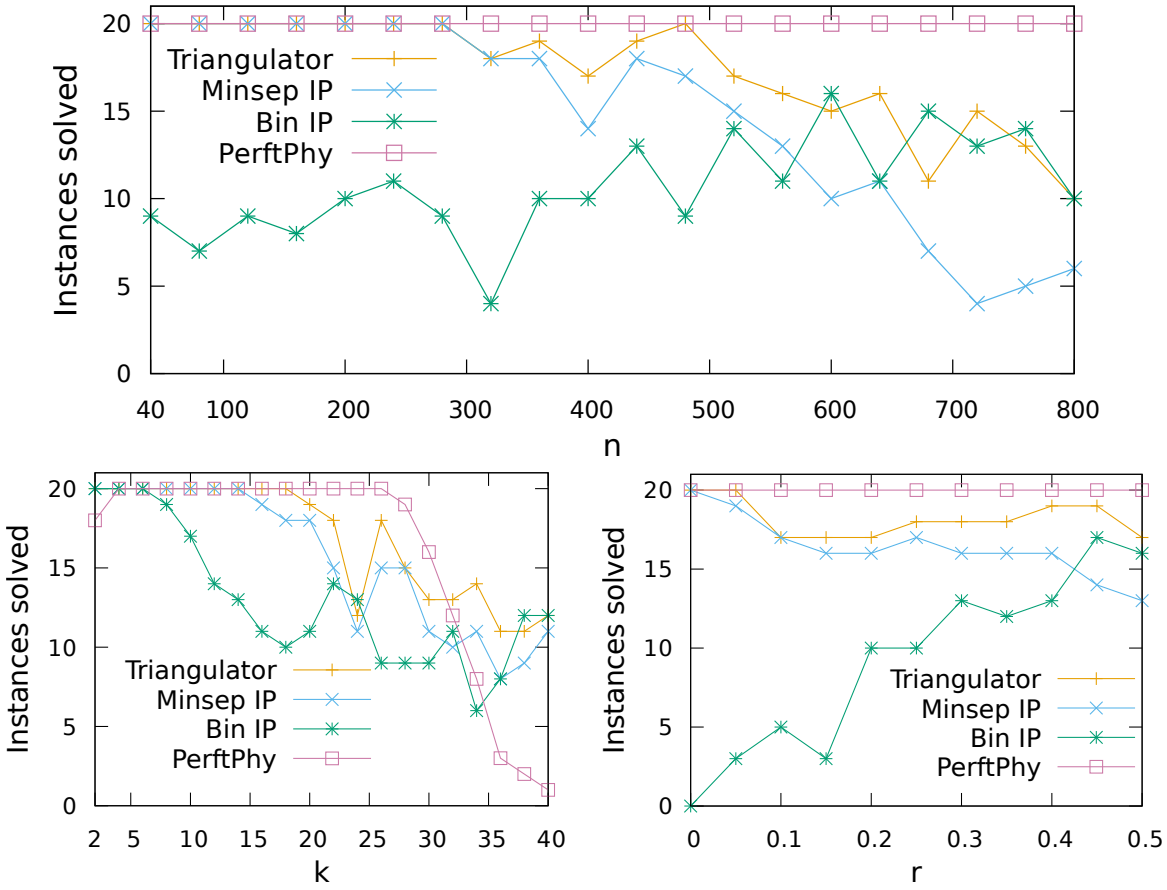

Figure 6.4: Number of solved instances out of the 20 generated for each combination of parameters on perfect phylogeny.

for producing non-trivial instances: in the three sets, 49%, 41%, and 47% of the solved instances admitted perfect phylogeny, respectively.

Figure 6.4 shows the results of the three experiments. PerftPhy is the best-performing algorithm overall, solving 939 instances out of the 1020 generated, with Triangulator second, solving 882 instances. The performance of Triangulator decreases with increasing n and k but stays fairly constant with increasing r. PerftPhy dominates other approaches in all cases except when k is high. This is likely explained by the fact that PerftPhy is based on an algorithm whose time complexity is exponential in k but polynomial in other parameters (assuming no missing data) [26, 65].

6.7.2 Binary Maximum Compatibility

For binary maximum compatibility, we do a similar scalability experiment with generated instances as for perfect phylogeny. In binary maximum compatibility, the arity $k = 2$ of the characters is fixed. We consider an additional parameter p, the probability of missing data. This is because if $p = 0$, then the binary maximum compatibility is equivalent to the

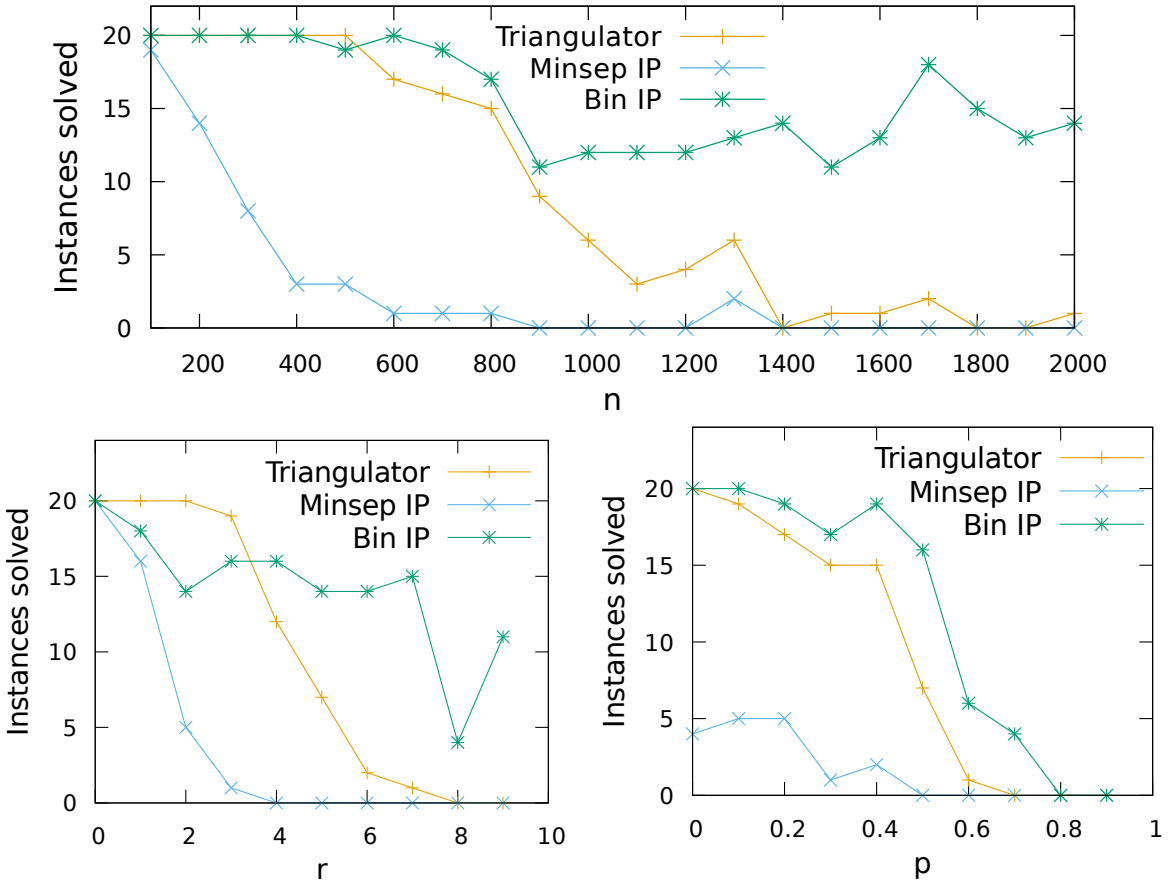

Figure 6.5: Number of solved instances out of the 20 generated for each combination of parameters on binary maximum compatibility.

maximum clique problem [29], so having $p > 0$ makes the problem phylogenetics-specific. We chose the parameters $n = m = 1000$, $r = 5$, and $p = 0.5$ as the baseline and generated three instance sets with ms. In the first set, the values of $n = m$ vary from 100 to 2000. In the second set, the value of r varies from 0 to 9. In the third set, the value of p varies from 0 to 0.9. For each considered combination of parameters we generated 20 instances. The median percentages of compatible characters in maximum compatible subsets in solved instances in these three sets are 77%, 89%, and 73%, respectively.

Figure 6.5 summarizes the results of the three experiments. Bin IP solves the most instances, solving 576 of the total 800 instances, with Triangulator second, solving 376 instances. Bin IP outperforms Triangulator especially on instances with $n \geq 1000$ and $r \geq 5$. On other instances, Bin IP has a comparable performance to Triangulator, both outperforming Minsep IP by a large margin.

6.7.3 Multi-state Maximum Compatibility

On multi-state maximum compatibility, we first compare the implementations on 5 real-world instances. The sources of these instances are Chinese dialects [84], Indo-European Languages [90], Mammal mitochondrial sequences [58] and Alcataenia [59]. Table 6.7 reports the results of the experiment on these instances and some properties of the instances. Triangulator and Bin IP solved all of the instances, and Minsep IP timed out on the "indo" instance.

As the second experiment on multi-state maximum compatibility we compare the scalability of the implementations on generated data similarly as on perfect phylogeny and on binary maximum compatibility. For multi-state maximum compatibility, we chose the values $n = m = 200$, $k = 20$, and $r = 2$ as the baseline and generated three instance sets with ms. In the first set, the values of $n = m$ vary from 20 to 400. In the second set, the value of k varies from 2 to 40. In the third set, the value of r varies from 0 to 3.9. The median percentages of compatible characters in maximum compatible subsets in solved instances in these sets are 83%, 77%, and 89%.

Figure 6.6 summarizes the results of these experiments. Triangulator outperforms the other approaches on scalability on all parameters, solving in total 682 of the 1200 instances. Triangulator dominates the other implementations on all datapoints except on $r = 3.6$, where Bin IP solves a couple of instances more.

As a third experiment on multi-state maximum compatibility, we compare Triangulator to Minsep IP and Bin IP in additional generated instances, with all parameter combinations from $n = m = \{50, 100, \ldots, 400\}$, $k = \{4, 10, 20, 40\}$ and $r = \{0, 1, 2, 4\}$. For each combination, 5 instances were generated. The median percentage of compatible characters in the solved instances is 90%.

Figure 6.7 summarizes the results of the third experiment, comparing directly the running times of each instance between Triangulator and Bin IP and Triangulator and Minsep IP. Figure 6.7 highlights the fact that Triangulator outperforms Bin IP especially on larger values of k and Minsep IP on larger values of r.

Table 6.7: Number of taxa, number of characters, arity of characters, the size of the maximum compatible subset of characters, and the runtimes of the implementations on real-world maximum compatibility instances.

Instance	n	m	k	comp	Approach		
					Triangulator	Minsep IP	Bin IP
alcataenia	9	14	3	9	0.06	0.11	0.28
chinese	7	15	5	9	0.08	0.07	0.28
indo	24	282	22	269	2033.05	TO	1.76
indo-pp	8	21	5	15	0.04	0.26	0.13
mammals	7	195	4	117	0.88	2.62	0.25

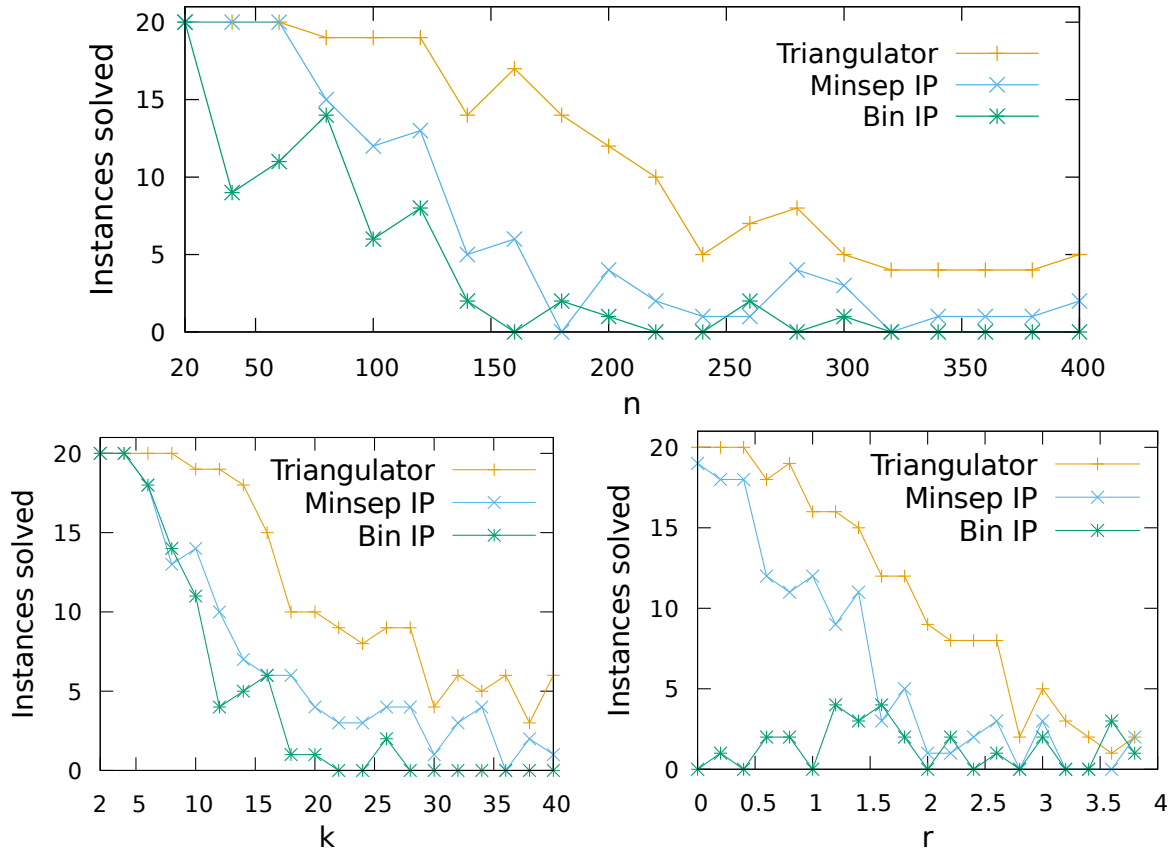

Figure 6.6: Number of solved instances out of the 20 generated for each combination of parameters on multi-state maximum compatibility.

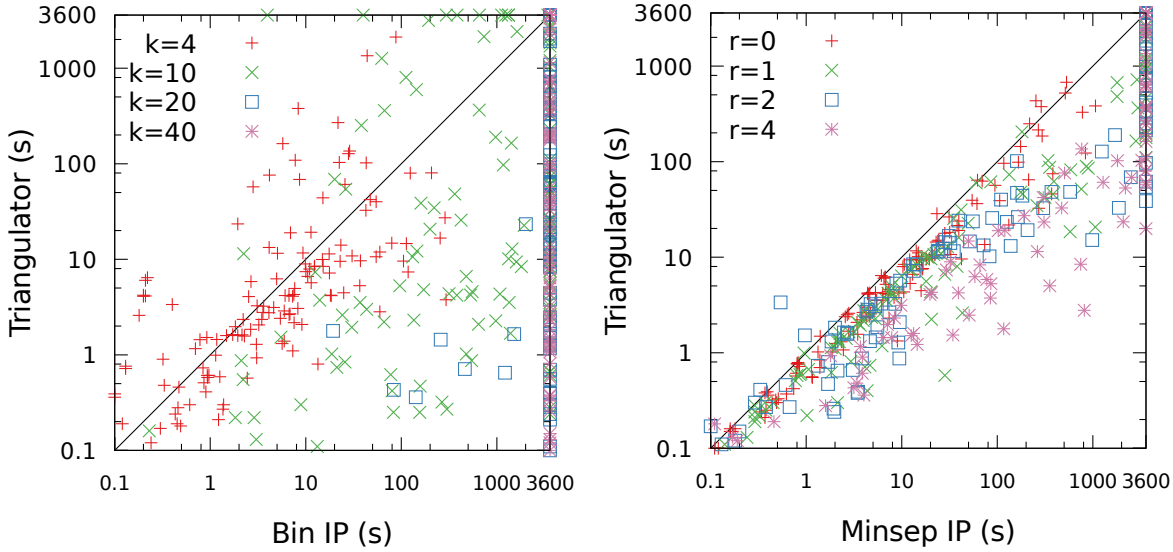

Figure 6.7: Comparison of the performance of Triangulator to the performance of Bin IP and Minsep IP on multi-state maximum compatibility.

Table 6.8: Number of instances solved on treelength, treewidth, and minimum fill-in, and the number of treelength instances solved in the preprocessing phase.

Instance set	#Instances	Approach			
		Triang TL	Triang TW	Triang MF	Triang TL PP
DIMACS	80	59	36	36	56
Named graphs	150	125	112	106	68
Total	230	184	148	142	124

6.8 Treelength

We are not aware of any other implementations for exact computing of treelength. Despite this, we experimented on computing treelength on the DIMACS graphs and the named graphs. Table 6.8 reports the number of instances solved on treelength, including also the numbers of instances solved on treewidth and minimum fill-in and the number of instances solved on treelength solely by the preprocessing phase.

Triangulator is able to solve more instances on treelength than on treewidth or minimum fill-in, especially in the DIMACS dataset. The explanation for this, offered by Table 6.8, is that most of the instances are solved for treelength in the preprocessing phase. Recall that in treelength preprocessing, the greatest lower bound that Triangulator is able to obtain is 2. Therefore, all instances that are solved by the preprocessing have treelength at most 2. Triangulator solves all instances with treelength 1 by recognizing if the graph is chordal and many of the instances with treelength 2 by first recognizing that the graph is not chordal and then finding a triangulation with treelength 2 with LB-Triang. Triangulator solves 110 of the 114 solved instances with treelength 2 solely by preprocessing.

Figure 6.8 shows the distribution of treelengths in solved instances, except for one instance with treelength 34. Of the 184 solved instances, 170 have treelength at most 4. The instance with treelength 34 is the cycle of 100 vertices.

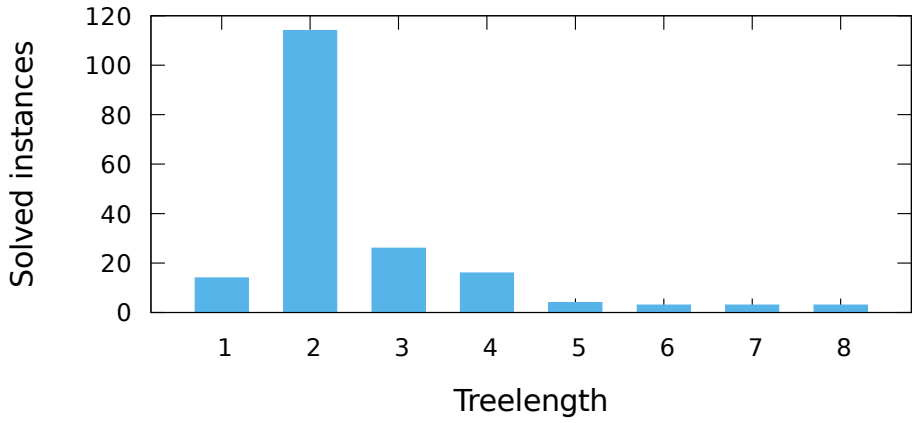

Figure 6.8: Number of solved treelength instances by treelength.

7 Analysis of the Implementation

Triangulator is a complex algorithm implementation whose performance depends on many factors, such as effectiveness of preprocessing, numbers of minimal separators and potential maximal cliques, and the performance of external libraries. In this chapter we further analyze data from the experiments reported in the previous chapter. We focus on understanding what kind of instances Triangulator is able to solve, why Triangulator is able to solve those instances, and how Triangulator could be improved to solve more instances. To make this chapter more concise we omit the analysis for total table size, binary maximum compatibility, and treelength. We remind the reader that the data on also those problems is available in the GitHub repository [69].

7.1 Treewidth

Triangulator solved 4826 of the 5441 treewidth instances. The flowchart of Triangulator on the treewidth instances in Figure 7.1 displays in which phases of Triangulator the instances were solved and in which phases Triangulator ran out of time or memory. For understanding the flowchart, recall Algorithm 4 of Chapter 5 representing Triangulator as pseudocode. In the flowchart we assume a model of Triangulator where the phases MS-ENUM, PMC-ENUM and BT-DP each happen exactly once, even though in reality there can be multiple sub-instances produced by the preprocessing phase, processed one by one in the solving phase. We assume that Triangulator only reached the phase in which it was terminated. For example, if Triangulator timed out in MS-ENUM, it is assumed to have never reached BT-DP even if another sub-instance was solved with BT-DP. We believe that this model is reasonably accurate for the purpose of identifying bottlenecks because the sub-instances are solved in non-decreasing order of the number of vertices.

Most of the instances were solved solely by the preprocessing phase. In 569 of the unsolved cases Triangulator was terminated in MS-ENUM, in 45 cases in PMC-ENUM, and in one case in the BT-DP phase. The fact that only one timeout occurred in BT-DP is not

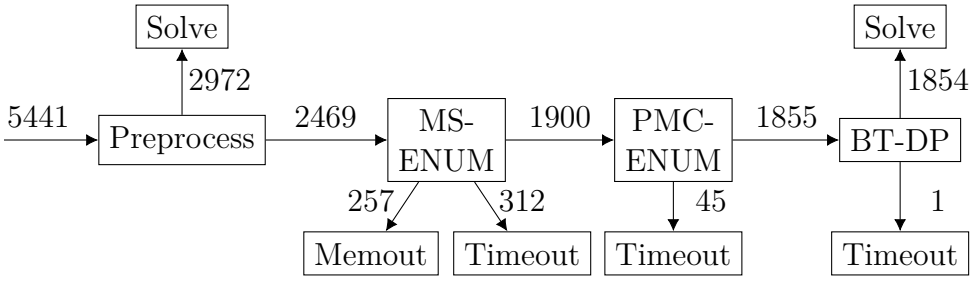

Figure 7.1: Number of treewidth instances solved in each phase of Triangulator and numbers of treewidth instances on which Triangulator ran out of time or memory.

surprising, since the runtime of BT-DP has only a small overhead on top of the size of the output of PMC-ENUM, while PMC-ENUM in general can have a large overhead over the size of its output. The fact that only 45 timeouts occurred in PMC-ENUM could be interpreted to indicate that the optimizations for PMC-ENUM are effective, enabling PMC-ENUM to be almost as fast as MS-ENUM. Another interpretation could be that there are not many instances with an intermediate number of minimal separators, i.e., most of the instances either have a very large number of minimal separators, causing a timeout or memout in MS-ENUM, or a relatively small number of minimal separators, enabling successful enumeration of PMCs.

To quantify the effect of preprocessing beyond the 2972 instances solved solely by preprocessing, we investigate the *kernel size* of the instances. The kernel size is the number of vertices in the largest sub-instance after preprocessing. Considering the 1854 instances that were solved after the preprocessing phase, 248 had at least 100 vertices but only 29 had kernel size at least 100. Figure 7.2 shows the percentage of solved instances among instances with each kernel size between 0 and 150, rounded to the nearest multiple of 10. Triangulator clearly performs better on instances with small kernel size than on instances with large kernel size. Most of the instances with kernel size at most 80 are solved, while most of the instances with kernel size at least 100 are not solved. The solved instance with the largest kernel has kernel size 1724, and the unsolved instance with the smallest kernel has kernel size 50. Considering all instances, the largest solved instance had 3,282 vertices before preprocessing and the smallest unsolved instance 50 vertices.

To analyze the effect of the number of minimal separators, Figure 7.3 shows treewidth instances as points determined by the kernel size and the number of enumerated minimal separators. The instances on which Triangulator was terminated in MS-ENUM appear to form a boundary in the figure, ranging from the unsolved instance with the largest number of enumerated minimal separators, having 913,767,685 enumerated minimal separators and kernel size 64, to an instance with 4,655,007 enumerated minimal separators and kernel size 2,809. The cases on which Triangulator was terminated in MS-ENUM are generally

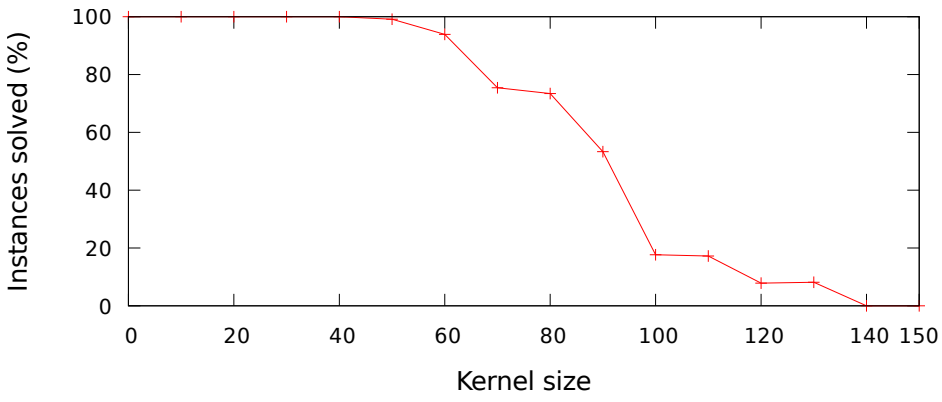

Figure 7.2: Percentage of solved treewidth instances among instances with each kernel size from 0 to 150. The kernel size of each instance is rounded to the nearest multiple of 10. Each datapoint is a ratio over at least 30 instances.

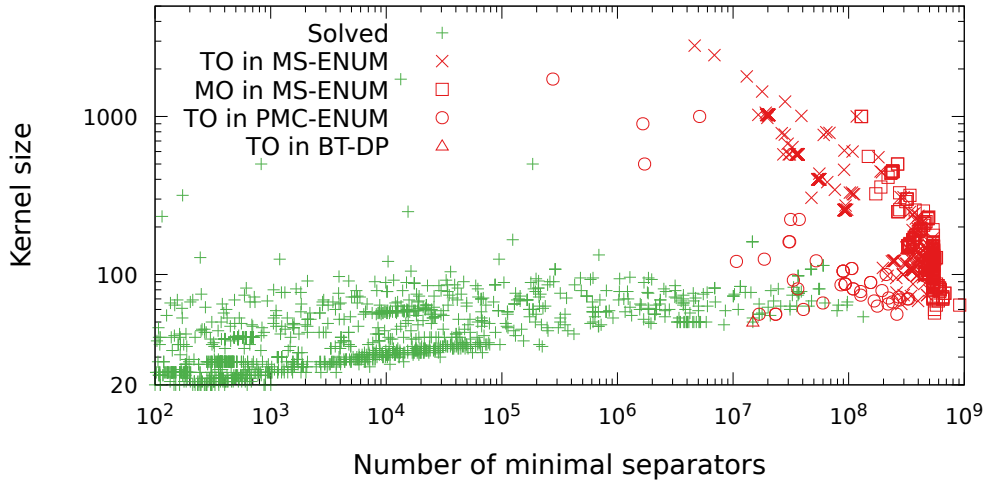

Figure 7.3: Kernel sizes and numbers of enumerated minimal separators in treewidth instances. Both axis are log scale. The instances are grouped by whether Triangulator solved them or ran out of time (TO) or memory (MO) in some phase.

quite close to this boundary, and the cases on which Triangulator was terminated in PMC-ENUM are closer to the origin. The solved instance with the largest number of minimal separators has 133,539,546 minimal separators and kernel size 54.

Figure 7.3 indicates that the performance of Triangulator depends on both the kernel size and the number of minimal separators. However, one may note that there are not many instances that have a large kernel and a small number of minimal separators, even though this is possible in principle. This might indicate that the preprocessing techniques for treewidth are effective in reducing the sizes of instances that have a small number of

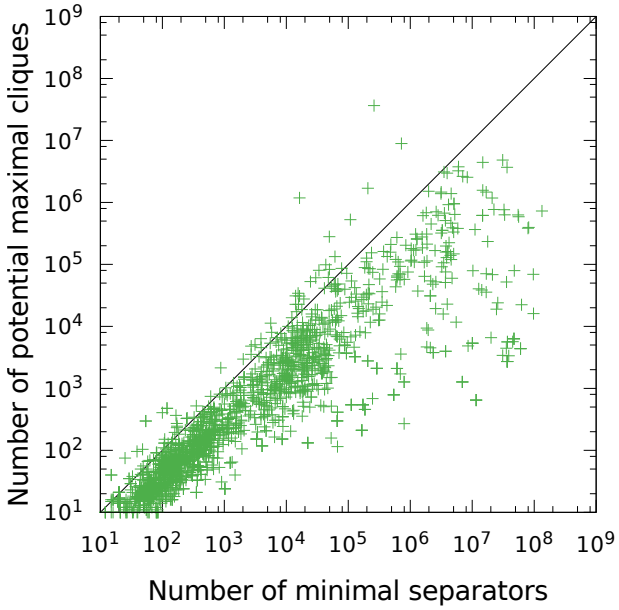

Figure 7.4: Numbers of enumerated minimal separators and potential maximal cliques in solved treewidth instances. Both axis are log scale.

minimal separators.

Figure 7.4 shows the numbers of enumerated minimal separators and PMCs in solved treewidth instances. The greatest number of potential maximal cliques in a solved instance is 36,536,076. Recall that on treewidth, Triangulator enumerates all minimal separators but only the PMCs Ω with $|\Omega| - 1 < ub$, where ub is an upper bound for treewidth. This seems to result in the number of enumerated PMCs being usually significantly smaller than the number of enumerated minimal separators. This fact could explain why Triangulator was terminated more often in MS-ENUM than in PMC-ENUM, suggesting that most of the enumerated minimal separators were larger than ub in size, and thus skipped in many parts of PMC-ENUM.

7.2 Minimum Fill-In

Triangulator solved 203 of the 330 minimum fill-in instances. The flowchart in Figure 7.5 displays in which phases of Triangulator the instances were solved and in which phases Triangulator ran out of time or memory. Triangulator solved 50 instances already in the preprocessing phase, but ran out of time on one instance in preprocessing, showing that the efficiency of the preprocessing techniques could be improved. The single instances on which Triangulator timed out in preprocessing has 35,588 vertices and 572,914 edges. Similarly as on treewidth, on minimum fill-in most of the timeouts and memouts occurred in MS-ENUM, with only one timeout in BT-DP. However, in minimum fill-in the difference between MS-ENUM and PMC-ENUM is not as large as in treewidth, suggesting that in minimum fill-in less minimal separators can be pruned via upper bounds in PMC-ENUM.

Triangulator solved 33 instances with kernel size at least 100 and 51 instances with at least 100 vertices. Figure 7.6 shows the percentage of solved instances among instances with each kernel size from 0 to 130, rounded to the nearest multiple of 10. The largest kernel size of a solved instance is 559, and the smallest kernel size of an unsolved instance is 50. Most instances with kernel size at most 50 are solved, but in general on minimum fill-in the kernel size does not appear to correlate as well with which instances Triangulator solves as on treewidth. We believe that this is because the preprocessing techniques for minimum fill-in are not as effective as for treewidth.

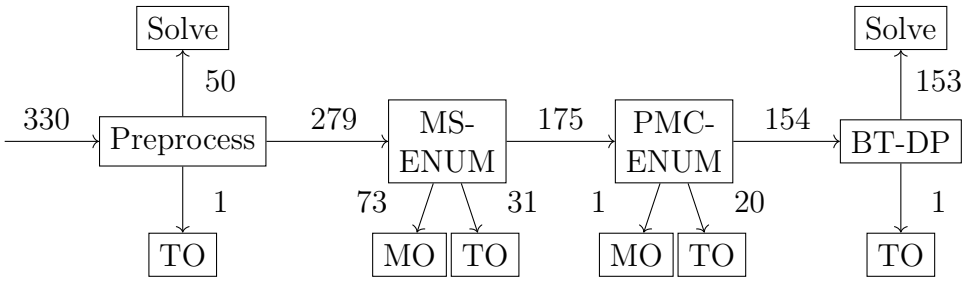

Figure 7.5: Number of minimum fill-in instances solved in each phase of Triangulator and numbers of minimum fill-in instances on which Triangulator ran out of time (TO) or memory (MO).

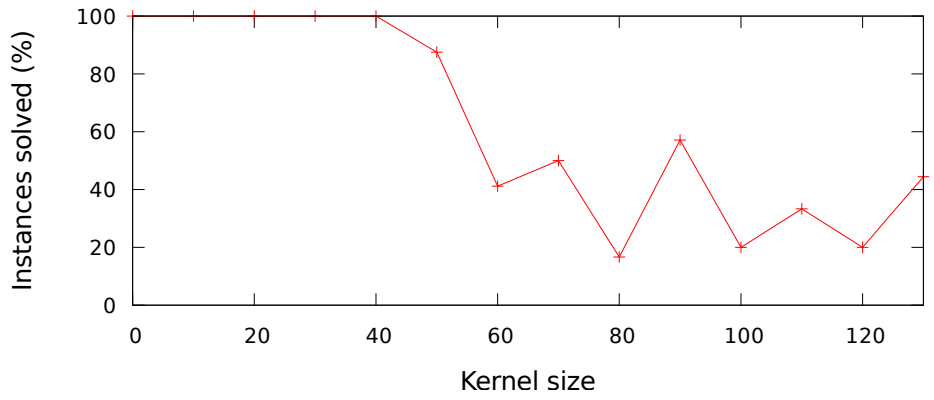

Figure 7.6: Percentage of solved minimum fill-in instances among instances with each kernel size from 0 to 130. The kernel size of each instance is rounded to the nearest multiple of 10. Each datapoint is a ratio over at least 6 instances.

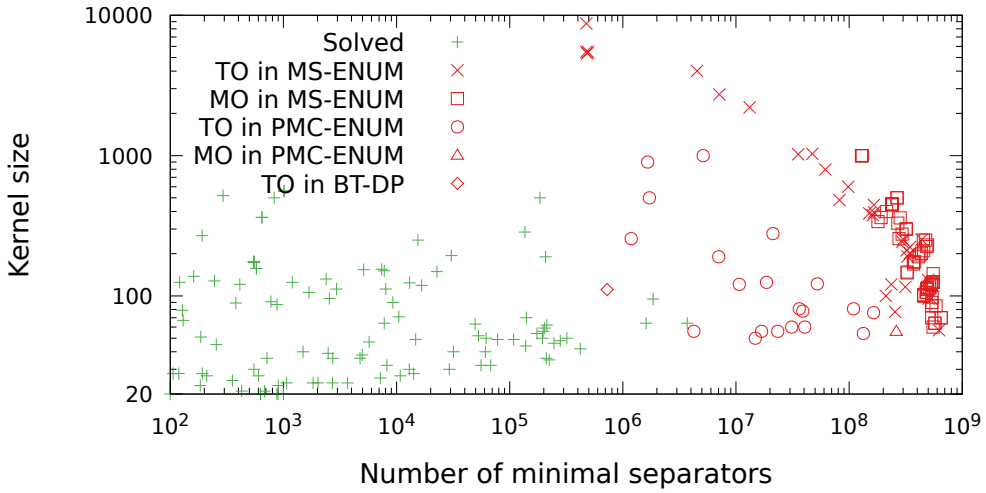

Figure 7.7: Kernel sizes and numbers of enumerated minimal separators in minimum fill-in instances. Both axis are log scale. The instances are grouped by whether Triangulator solved them or ran out of time (TO) or memory (MO) in some phase.

Figure 7.7 shows the kernel sizes and the numbers of enumerated minimal separators on minimum fill-in instances grouped by the status of Triangulator on them. The figure shows a similar boundary of instances on which Triangulator was terminated in MS-ENUM as the corresponding figure for treewidth. The numbers of enumerated minimal separators in instances on which Triangulator was terminated in MS-ENUM range from 474,748 to 650,670,407. Compared to treewidth, on minimum fill-in the figure shows a larger region consisting of instances on which Triangulator ran out of time or memory in PMC-ENUM. The largest solved instance in terms of the number of minimal separators has 3,701,564 minimal separators and kernel size 64. This indicates that on minimum fill-in a significant bottleneck of Triangulator is the PMC-ENUM phase, in contrast to treewidth on which Triangulator is able to solve instances with over 100 million minimal separators.

Figure 7.8 shows the numbers of enumerated minimal separators and PMCs on solved

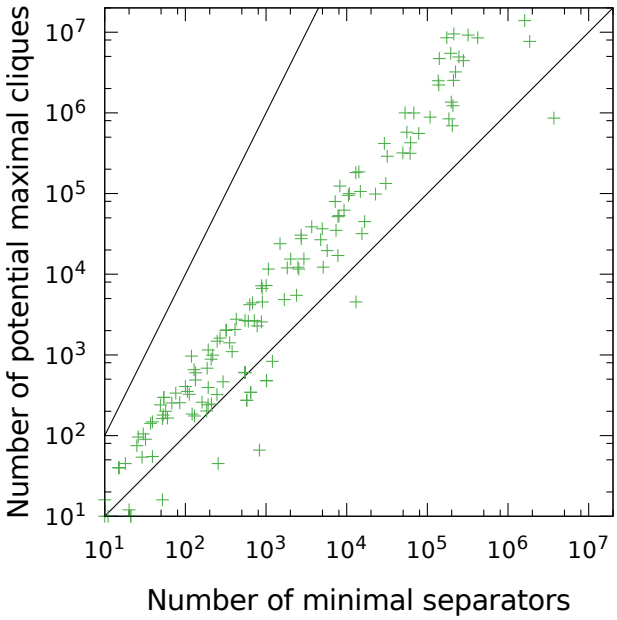

Figure 7.8: Numbers of enumerated minimal separators and potential maximal cliques on solved minimum fill-in instances. The polynomials x and x^2 are also shown. Both axis are log scale.

minimum fill-in instances. The number of enumerated PMCs is consistently higher than the number of enumerated minimal separators, appearing to almost form a line in the figure.[1] The slope of the line seems to be much smaller than what would result from the known quadratic bound between the numbers of PMCs and minimal separators.

7.3 Generalized Hypertreewidth

On generalized hypertreewidth, Triangulator solved 2544 of the 3070 instances. The flowchart in Figure 7.9 displays in which phases of Triangulator the instances were solved and in which phases Triangulator ran out of time or memory. The edge cover computation phase is included in the flowchart between the PMC-ENUM and BT-DP phases. The

[1]Note that log-log scale transforms a polynomial bx^a to linear function $ax + b$.

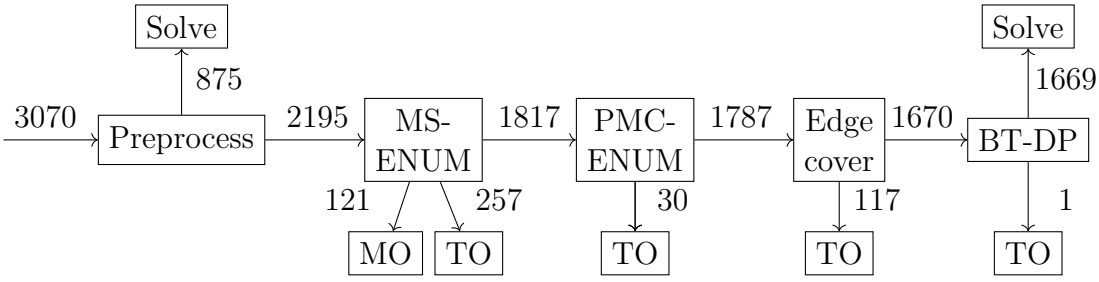

Figure 7.9: Number of generalized hypertreewidth instances solved in each phase of Triangulator and numbers of generalized hypertreewidth instances on which Triangulator ran out of time (TO) or memory (MO).

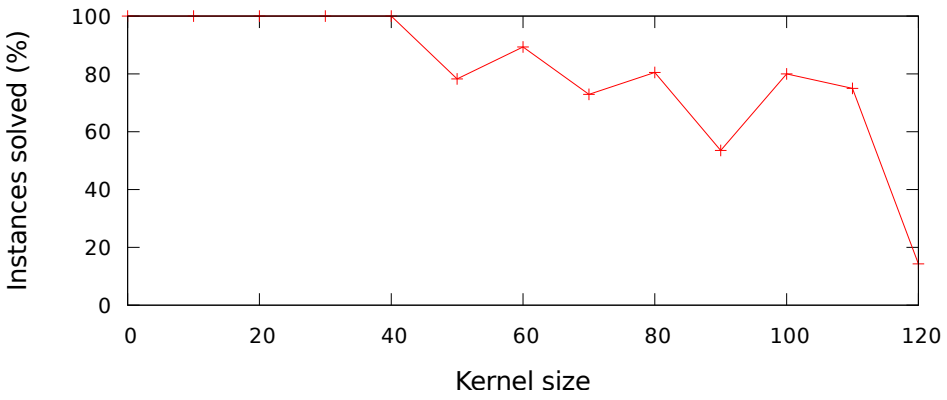

Figure 7.10: Percentage of solved generalized hypertreewidth instances among instances with each kernel size from 0 to 120. The kernel size of each instance is rounded to the nearest multiple of 10. Each datapoint is a ratio over at least 7 instances.

preprocessing phase solved 875 of the 2544 solved instances. Because the greatest lower bound for GHTW that the preprocessing is able to obtain is 2, these instances had either GHTW at most 2 or their primal graphs were chordal. Most of the timeouts and memouts occurred in the MS-ENUM phase, with also a significant number of timeouts in the edge cover phase. Similarly as on treewidth and minimum fill-in, only one timeout occurred in the BT-DP phase.

Figure 7.10 shows the percentage of solved instances among instances with each kernel size between 0 and 120, rounded to the nearest multiple of 10. Triangulator solved 60 instances with kernel size at least 100 and 140 instances with at least 100 vertices. The solved instance with the largest kernel had kernel size 887 and the solved instance with the largest number of vertices had 893 vertices. The unsolved instance with the smallest kernel had kernel size 50. The kernel size seems to not correlate as well with the percentage of solved instances as on treewidth and minimum fill-in. This may be due to the fact that the preprocessing techniques for GHTW are more limited than the preprocessing techniques for treewidth and minimum fill-in.

Figure 7.11 shows the kernel sizes and the numbers of enumerated minimal separators on GHTW instances grouped by the status of Triangulator on them. The figure shows a boundary consisting of instances on which Triangulator was terminated either due to timeout or memout in MS-ENUM, ranging from an instance with 650,678,091 enumerated minimal separators and kernel size 72 to an instance with 6,894,338 enumerated minimal separators and kernel size 2813. Towards the origin from this boundary, the figure shows a mixture of instances on which Triangulator timed out in PMC-ENUM, edge cover computation and in BT-DP. The fact on whether Triangulator timed out in the edge cover phase seems fairly independent on the number of minimal separators. The solved instance with the largest number of minimal separators has 58,306,891 minimal separators. The instances with the least number of minimal separators on which Triangulator timed out in the edge cover phase and the BT-DP phase have 32,940 and 14,956 minimal separators, respectively. This suggests that the number of PMCs in these instances could be

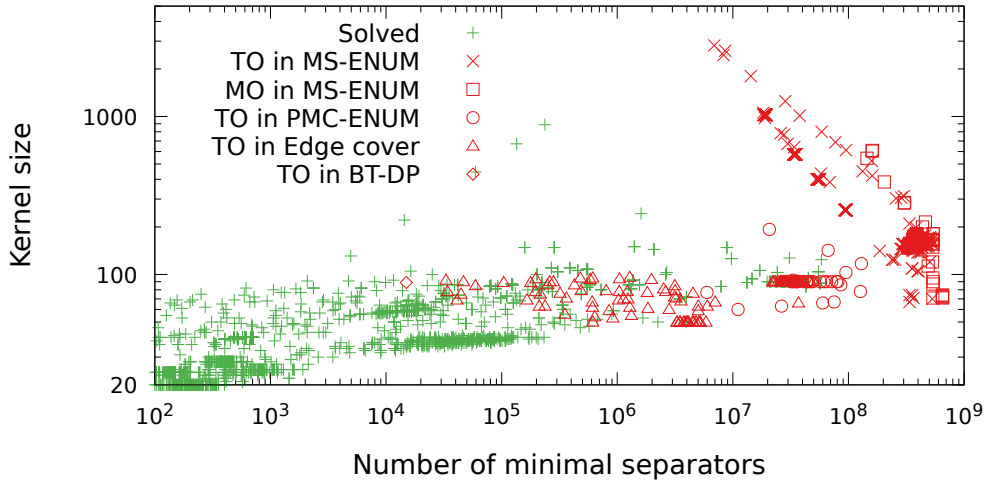

Figure 7.11: Kernel sizes and numbers of enumerated minimal separators in generalized hypertreewidth instances. Both axis are log scale. The instances are grouped by whether Triangulator solved them or ran out of time (TO) or memory (MO) in some phase.

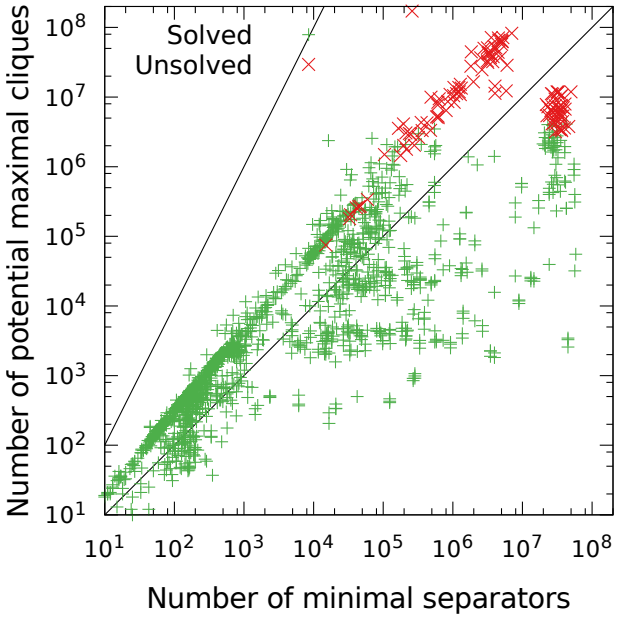

Figure 7.12: Numbers of enumerated minimal separators and potential maximal cliques on generalized hypertreewidth instances. The polynomials x and x^2 are also shown. Both axis are log scale.

significantly higher than the number of minimal separators.

Figure 7.12 shows the numbers of enumerated minimal separators and PMCs on GHTW instances grouped by whether Triangulator solved the instance. The unsolved cases correspond to timeouts in the edge cover and BT-DP phases. Recall that on GHTW, even if Triangulator does not prune all PMCs Ω with $f(\Omega) \geq ub$ in the PMC-ENUM phase, it uses heuristics to prune some of the PMCs before edge cover computation. The number of PMCs seems to explain the timeouts in edge cover computation better than the number of minimal separators, although there are a few outliers. The solved instance with the largest

number of enumerated PMCs has 4,095,897 PMCs. The unsolved instance on which Triangulator was terminated after PMC-ENUM with the least number of enumerated PMCs has 75,029 PMCs. Since the total number of PMCs in a graph G is at least $|\Delta(G)|/n$ [22], we can infer from Figure 7.12 that the pruning heuristic used in PMC-ENUM for GHTW is effective in reducing the number of enumerated PMCs.

7.4 Fractional Hypertreewidth

On fractional hypertreewidth Triangulator solved 2425 of the 3070 instances. The flowchart in Figure 7.13 displays in which phases of Triangulator the instances were solved and in which phases Triangulator ran out of time or memory. The preprocessing phase solved 753 instances. Triangulator does not include problem-specific preprocessing techniques for FHTW, so the instances solved in preprocessing are exactly the instances whose primal graphs are chordal. Note that this implies that on GHTW, 122 non-chordal instances were solved in preprocessing. Similarly as on GHTW, most of the timeouts and memouts occurred in MS-ENUM. However, on FHTW there are 118 more timeouts and 4 more memouts in PMC-ENUM, making PMC-ENUM a much more significant bottleneck than on GHTW. The number of timeouts in the edge cover phase is only two less compared to GHTW, even though in theory solving linear programs is more efficient than solving integer programs.

Figure 7.14 shows the percentage of solved instances among instances with each kernel size from 0 to 120, rounded to the nearest multiple of 10. Triangulator solved 16 instances with kernel size at least 100 and 46 instances with at least 100 vertices. The unsolved instance with the smallest kernel has kernel size 50 and the solved instance with the largest kernel has kernel size 221. The kernel size seems more significant factor in FHTW than in GHTW. There are 141 instances that Triangulator is able to solve for GHTW but not for FHTW, and 94 of those have kernel size at least 90.

Figure 7.15 shows the kernel sizes and the numbers of enumerated minimal separators on FHTW instances grouped by the status of Triangulator on them. The instances on which Triangulator was terminated in MS-ENUM form a similar boundary as on GHTW, but the

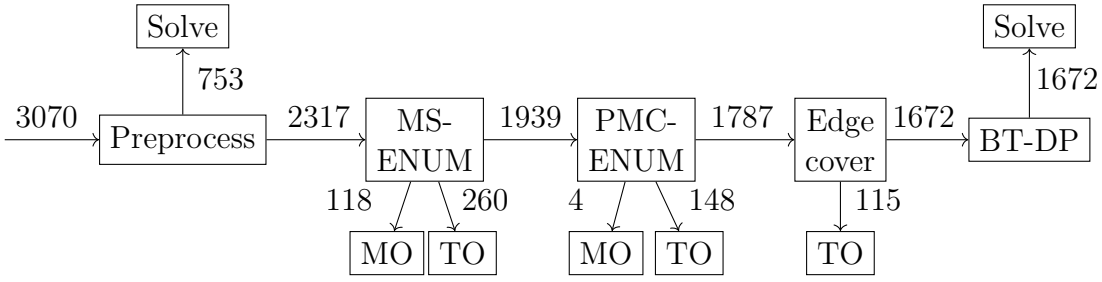

Figure 7.13: Number of fractional hypertreewidth instances solved in each phase of Triangulator and numbers of fractional hypertreewidth instances on which Triangulator ran out of time (TO) or memory (MO).

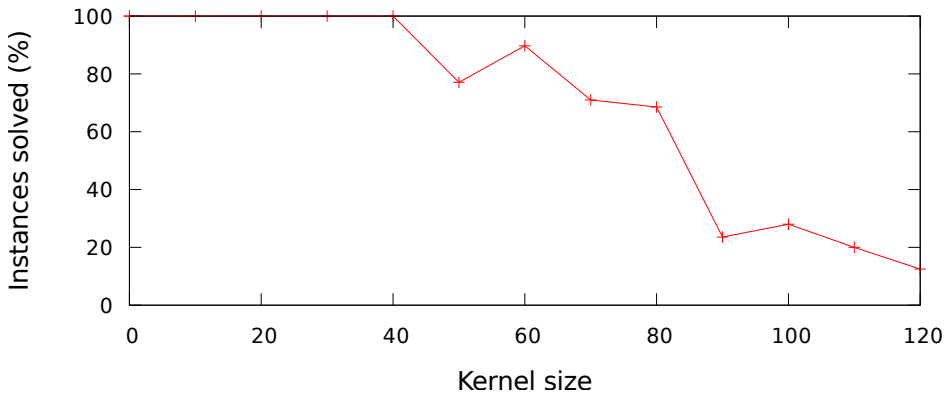

Figure 7.14: Percentage of solved fractional hypertreewidth instances among instances with each kernel size from 0 to 120. The kernel size of each instance is rounded to the nearest multiple of 10. Each datapoint is a ratio over at least 8 instances.

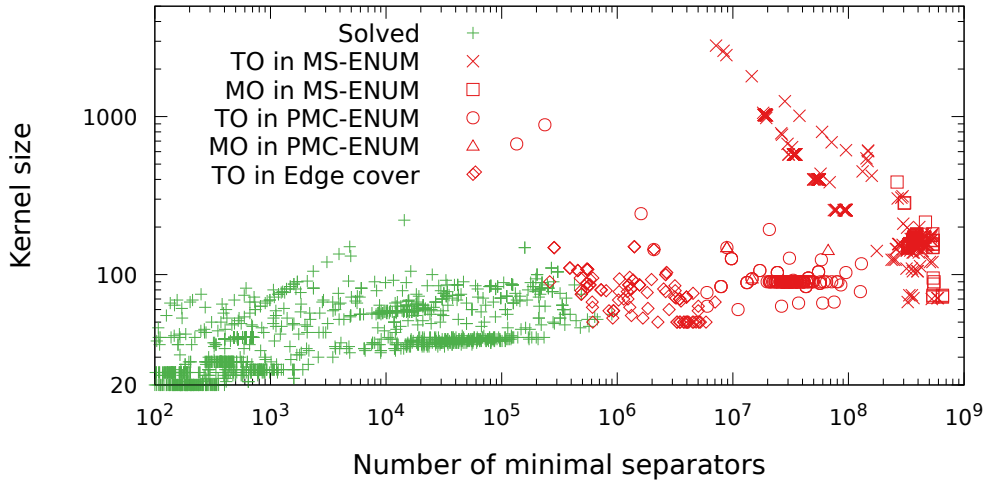

Figure 7.15: Kernel sizes and numbers of enumerated minimal separators in fractional hypertreewidth instances. Both axis are log scale. The instances are grouped by whether Triangulator solved them or ran out of time (TO) or memory (MO) in some phase.

interior of the figure is vastly different to the corresponding figure for GHTW. No instances with more than 1,000,000 minimal separators were solved on FHTW. The solved instance with the largest number of minimal separators has 898,258 minimal separators and kernel size 56, and the unsolved instance with the least number of minimal separators has 59,341 minimal separators and kernel size 446. This is a significant difference to GHTW, on which instances with more than 50 million minimal separators were solved.

Figure 7.16 shows the numbers of enumerated minimal separators and PMCs in FHTW instances. The greatest number of PMCs in a solved instance is 23,079,748 and the smallest number of PMCs in an unsolved instance past the PMC-ENUM phase is 3,326,577. The numbers of minimal separators in these instances are 488,785 and 478,591, respectively. On FHTW, Triangulator does not prune the PMCs in any way in PMC-ENUM, so the figure shows the true relation between the number of minimal separators and the number of

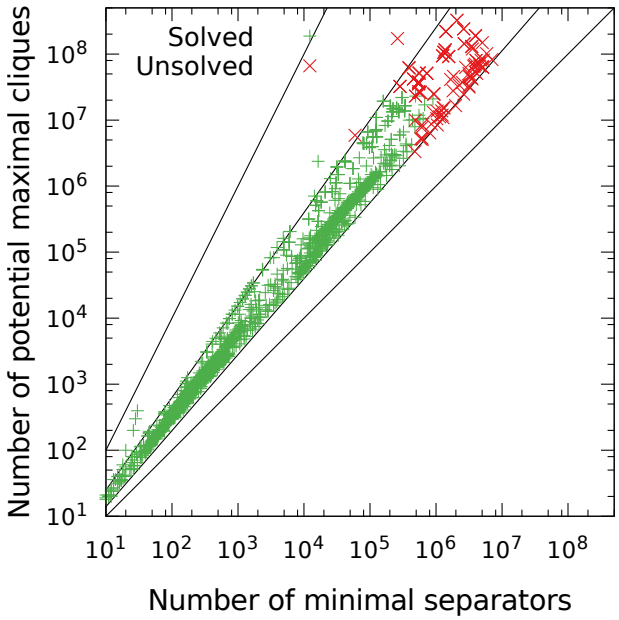

Figure 7.16: Numbers of enumerated minimal separators and potential maximal cliques on fractional hypertreewidth instances. The polynomials x, $x^{1.15}$, $x^{1.4}$ and x^2 are also shown. Both axis are log scale.

PMCs of the instances. All of the instances have more PMCs than minimal separators, and 92% of the 1787 instances satisfy $|\Delta(G)|^{1.15} \leq |\Pi(G)| \leq |\Delta(G)|^{1.4}$. The subquadraticity of the upper bound motivates asking whether there is an upper bound of form $|\Delta(G)|^c poly(n)$ where $c < 2$ for the number of PMCs of a graph G.

7.5 Perfect Phylogeny

Triangulator solved 882 of the 1020 perfect phylogeny instances. The flowchart in Figure 7.17 displays in which phases of Triangulator the instances were solved and in which phases Triangulator ran out of time or memory. The preprocessing phase solved 263 instances. Of the instances solved in preprocessing, 22 admit a perfect phylogeny and 241 do not. The instances that do not admit a perfect phylogeny were solved with the degeneracy heuristic. Triangulator did not run out of memory on any perfect phylogeny instance. The

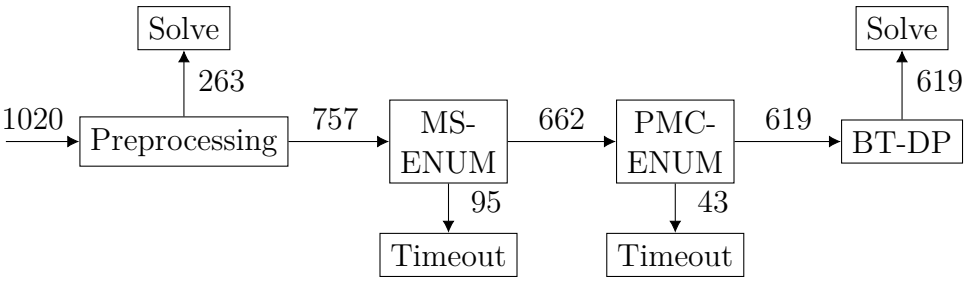

Figure 7.17: Number of perfect phylogeny instances solved in each phase of Triangulator and numbers of perfect phylogeny instances on which Triangulator ran out of time.

timeouts happened only in the MS-ENUM and PMC-ENUM phases, with 95 timeouts in MS-ENUM and 43 timeouts in PMC-ENUM.

The graphs in phylogenetics are in general larger than in other problems. The median kernel size among solved instances in perfect phylogeny is 1186. Figure 7.18 shows the percentage of solved instances for each kernel size between 0 and 4600, rounded to the nearest multiple of 100. The smallest kernel size of an unsolved instance is 2428 and the largest kernel size of a solved instance is 5737. Overall, the largest solved instance in terms of total number of vertices had 12,310 vertices.

Figure 7.19 shows the kernel sizes and the numbers of enumerated minimal separators in perfect phylogeny instances grouped by the status of Triangulator on them. The figure is vastly different to the corresponding figures for the already discussed problems. There is a cluster of instances on which Triangulator timed on in MS-ENUM and which have less than

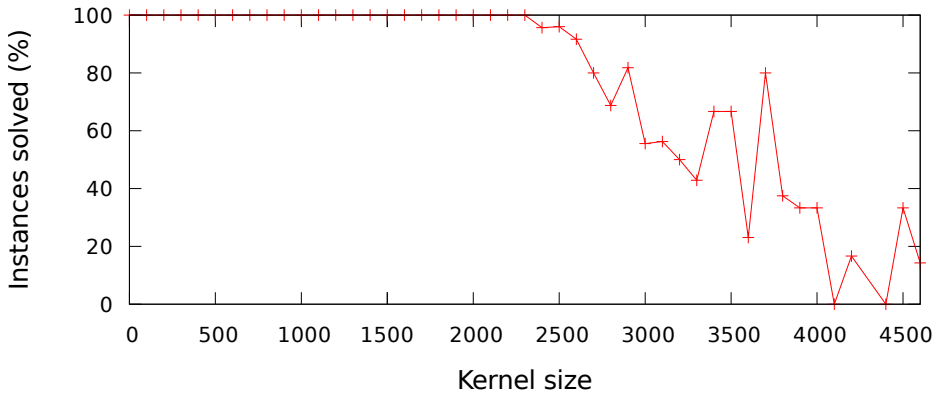

Figure 7.18: Percentage of solved perfect phylogeny instances among instances with each kernel size from 0 to 4600. The kernel size of each instance is rounded to the nearest multiple of 100. Each datapoint is a ratio over at least 5 instances.

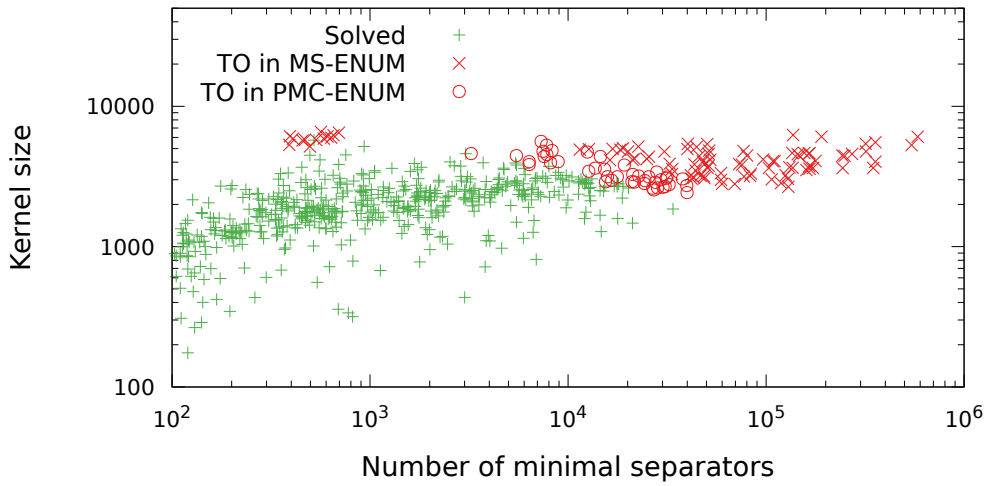

Figure 7.19: Kernel sizes and numbers of enumerated minimal separators in perfect phylogeny instances. Both axis are log scale. The instances are grouped by whether Triangulator solved them or ran out of time (TO) in some phase.

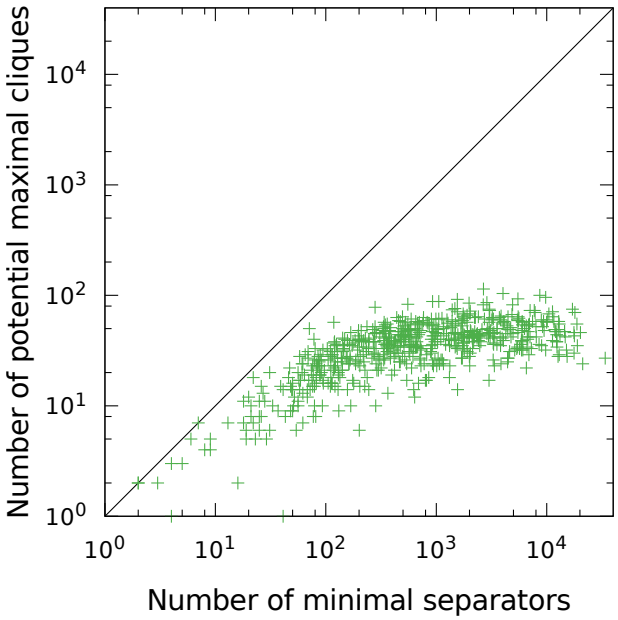

Figure 7.20: Numbers of enumerated minimal separators and potential maximal cliques on perfect phylogeny instances. Both axis are log scale.

1000 minimal separators. The instance with the smallest number of minimal separators on which Triangulator timed out in MS-ENUM has 391 enumerated minimal separators and 5315 vertices, though we have to remark that due to certain implementation details, the reported numbers of enumerated minimal separators are not accurate for instances on which Triangulator was terminated in MS-ENUM after enumerating less than 10,000 minimal separators.

Figure 7.20 shows the numbers of enumerated minimal separators and PMCs on perfect phylogeny instances. Recall that on perfect phylogeny, Triangulator prunes a PMC if it contains two vertices corresponding to a same character. By Figure 7.20, this pruning appears to be effective, leaving the numbers of enumerated PMCs to be two magnitudes smaller than the numbers of enumerated minimal separators. The instance with the largest number of enumerated PMCs has 114 PMCs, 2677 minimal separators, and kernel size 2474.

7.6 Multi-State Maximum Compatibility

Triangulator solved 1152 of the 1845 multi-state maximum compatibility instances. The previous implementation of Triangulator, used in the experiments of [72] solved 1012 instances on the same instance set, running on the same computers with a 2-hour time limit. The previous implementation used the same MaxSAT solver, so we believe that the improvement was mostly due to the new PMC-ENUM algorithm.

The flowchart in Figure 7.21 displays in which phases of Triangulator the instances were solved and in which phases Triangulator ran out of time or memory. The final phase for

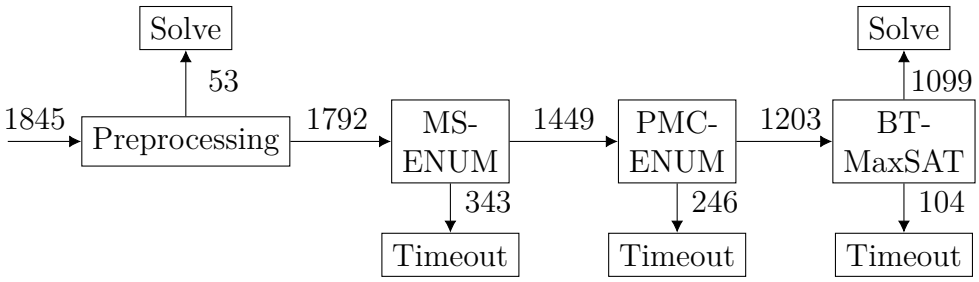

Figure 7.21: Number of multi-state maximum compatibility instances solved in each phase of Triangulator and numbers of multi-state maximum compatibility instances on which Triangulator ran out of time.

multi-state maximum compatibility is BT-MaxSAT, including both building the MaxSAT instance and solving it with the MaxHS [28] MaxSAT solver. The BT-MaxSAT phase is a more significant bottleneck than the BT-DP phase of the already discussed problems. Triangulator timed out in BT-MaxSAT on 104 instances. However, one may argue that 104 timeouts is not that much when taking into account that MaxSAT is an NP-hard problem and that more timeouts occurred in both MS-ENUM and PMC-ENUM.

The sizes of instances in multi-state maximum compatibility are larger than in non-phylogeny problems but not as large as in perfect phylogeny. The median kernel size among solved instances is 725. Figure 7.22 shows the percentage of solved instances for kernel sizes between 0 and 2500, rounded to the nearest multiple of 100. The unsolved instance with the smallest kernel has kernel size 327 and the solved instance with the largest kernel has kernel size 3123. The percentage of solved instances appears to correlate well with the kernel size. Triangulator solves most of the instances with kernel size at most 1000 and does not solve most of the instances with kernel size at least 1200.

Figure 7.23 shows the kernel sizes and the numbers of enumerated minimal separators in multi-state maximum compatibility instances grouped by the status of Triangulator

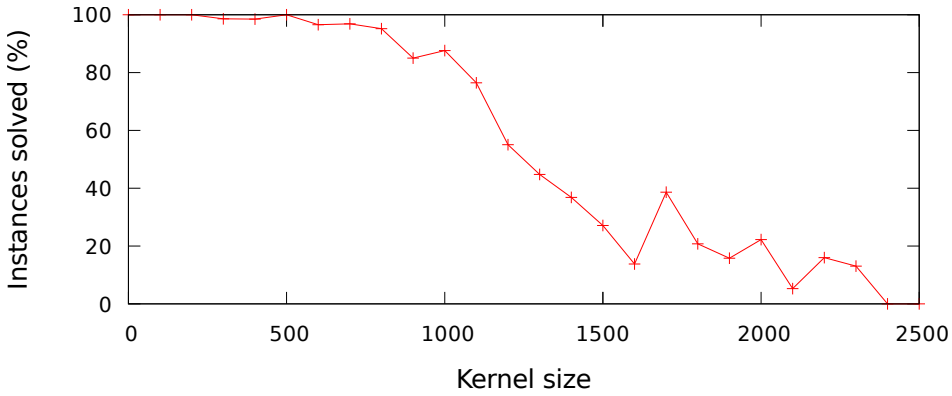

Figure 7.22: Percentage of solved multi-state maximum compatibility instances among instances with each kernel size from 0 to 2500. The kernel size of each instance is rounded to the nearest multiple of 100. Each datapoint is a ratio over at least 10 instances.

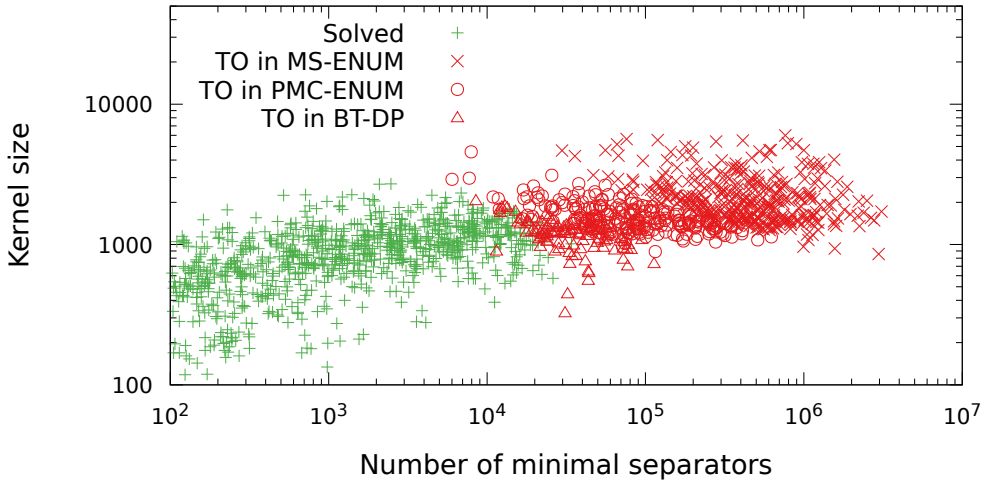

Figure 7.23: Kernel sizes and numbers of enumerated minimal separators in multi-state maximum compatibility instances. Both axis are log scale. The instances are grouped by whether Triangulator solved them or ran out of time (TO) in some phase.

on them. The instance with the largest number of enumerated minimal separators has 3,088,150 minimal separators and kernel size 1717. The solved instance with the largest number of minimal separators has 34,071 minimal separators and kernel size 1012. The unsolved instance with the least number of enumerated minimal separators has 6007 enumerated minimal separators and kernel size 2913. The number of minimal separators appears to correlate well with which instances Triangulator solves.

Figure 7.24 shows the numbers of enumerated minimal separators and PMCs in multi-state

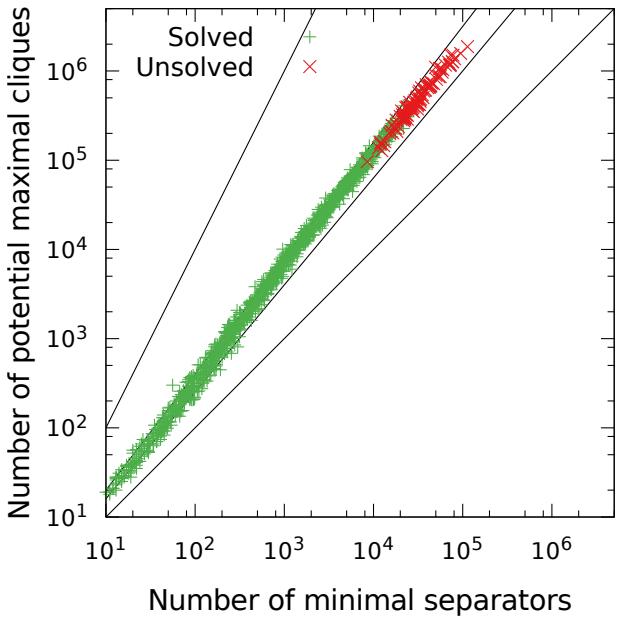

Figure 7.24: Numbers of enumerated minimal separators and potential maximal cliques on multi-state maximum compatibility instances. The polynomials x, $x^{1.2}$, $x^{1.3}$ and x^2 are also shown. Both axis are log scale.

maximum compatibility instances. Similarly as in FHTW, no PMCs are pruned in the PMC-ENUM phase, so the figure shows the true relation between the numbers of minimal separators and PMCs of the graphs. The unsolved instances in the figure correspond to instances on which Triangulator timed out in the BT-MaxSAT phase. The timeouts of that phase seem to be consistently explained by the number of PMCs. The solved instance with the largest number of PMCs has 609,002 PMCs and the unsolved instance past the PMC-ENUM phase with the least number of PMCs has 96,086 PMCs. The relation of the number of PMCs and the number of minimal separators appears to follow a line in the figure, with most of the instance having the number of PMCs between $|\Delta(G)|^{1.2}$ and $|\Delta(G)|^{1.3}$. We note that this phenomenon may not be a general phenomenon of graphs but rather be explained by the properties of the ms instance generator for the phylogeny instances.

8 Conclusions

In this thesis we focused on the Bouchitté–Todinca algorithm for finding optimal tree decompositions of graphs, with multiple different notions of optimality. We improved the potential maximal clique enumeration phase of the BT algorithm, reducing the time complexity of the phase by a factor of n and resulting in a faster implementation. We introduced an adaptation of the BT algorithm for the maximum compatibility problem of multi-state phylogenetic characters. We implemented the BT algorithm for seven problems arising from different areas of computer science. Our implementation, called Triangulator, outperforms other implementations in exact computing of minimum fill-in, generalized hypertreewidth, fractional hypertreewidth, total table size, maximum compatibility of multi-state phylogenetic characters, and treelength.

While in theory the potential applications of the BT algorithm are well-known, our work is one of the first to experimentally evaluate the BT algorithm, and to our knowledge the first to extensively evaluate the BT algorithm on other optimization problems than treewidth and minimum fill-in. One of the most common approaches for finding optimal tree decompositions in practice has been via elimination orderings, used for example by QuickBB, FraSMT and EDFS [38, 48, 77]. Our implementation of the BT algorithm consistently outperforms these implementations, suggesting that the BT algorithm is the superior of these two generic approaches. In the problems on which alternative implementations outperform our implementation, they are either based on the BT algorithm themselves, as in treewidth, or highly problem-specific, as in perfect phylogeny and maximum compatibility of binary phylogenetic characters.

Future research. In this thesis we applied the BT algorithm in the context of finding tree decompositions. Beyond tree decompositions, Fomin et al. have formulated a framework showing that problems expressed as finding a maximum size induced subgraph with treewidth t can be solved in $O(|\Pi(G)|n^{4+t})$ time, given the set $\Pi(G)$ of potential maximal cliques of the input graph G [43]. This framework includes, for example, the maximum independent set problem (with $t = 0$), the longest induced path problem (with $t = 1$), and the minimum feedback vertex set problem (with $t = 1$) [43]. To the best of our knowledge, there are no implementations of instances of this framework. Implementing the algorithm suggested by this framework for some application and investigating its applicability in practice is an interesting direction for future research.

In addition to applying the BT algorithm to more problems, another natural future research direction is to improve the efficiency of the algorithm. For improving the PMC-ENUM phase, we note that enumerating potential maximal cliques in $O(|\Pi(G)|\text{poly}(n))$ time has been explicitly stated as an open problem in 2006 [20] and to our knowledge is still open. A practical algorithm with such time complexity could be used to improve Triangulator, turning some of the timeouts in the PMC-ENUM phase into solved cases.

Based on our experiments, a more significant bottleneck than PMC-ENUM for Triangulator is the sheer number of minimal separators, often filling the 64 GB of memory before reaching the 1-hour time limit in our experiments. Recall that the number of PMCs is at least the number of minimal separators divided by the number of vertices [22], so these instances also have a large number of PMCs. Therefore, a practically motivated direction would be to reduce the number of objects that need to be enumerated. This direction has been explored in the contexts of treewidth and generalized hypertreewidth by us [70] and in the contexts of treewidth and minimum fill-in as the "positive instance driven" approach of Tamaki et al. [33, 98, 99]. In the experiments of this thesis, the positive instance driven approach outperforms Triangulator in computing treewidth, but Triangulator outperforms it in computing minimum fill-in. A practical algorithm that would enumerate in a linear-output-sensitive manner the PMCs whose associated cost is less than a given bound would likely be an improvement over all of these approaches [70, 99].

Acknowledgements

This work has been financially supported by Academy of Finland under grants 312662 and 322869. Computational resources were provided by Finnish Grid and Cloud Infrastructure [37].

Bibliography

[1] Christopher R. Aberger, Andrew Lamb, Susan Tu, Andres Nötzli, Kunle Olukotun, and Christopher Ré. EmptyHeaded: A relational engine for graph processing. *ACM Transactions on Database Systems*, 42(4):20:1–20:44, 2017.

[2] Muad Abu-Ata and Feodor F. Dragan. Metric tree-like structures in real-world networks: An empirical study. *Networks*, 67(1):49–68, 2016.

[3] Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *European Journal of Combinatorics*, 28(8):2167–2181, 2007.

[4] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

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

[6] Max Bannach and Sebastian Berndt. Positive-instance driven dynamic programming for graph searching. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R. Salavatipour, editors, *Proceedings of the 16th International Symposium on Algorithms and Data Structures*, volume 11646 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 2019.

[7] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 825–885. IOS Press, 2009.

[8] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.

[9] Jeremias Berg and Matti Järvisalo. SAT-based approaches to treewidth computation: An evaluation. In *Proceedings of the 26th International Conference on Tools with Artificial Intelligence*, pages 328–335. IEEE Computer Society, 2014.

[10] David Bergman, Carlos H. Cardonha, Andre A. Cire, and Arvind U. Raghunathan. On the minimum chordal completion polytope. *Operations Research*, 67(2):532–547, 2019.

[11] Anne Berry, Jean R. S. Blair, Pinar Heggernes, and Barry W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287–298, 2004.

[12] Anne Berry, Jean-Paul Bordat, and Olivier Cogis. Generating all the minimal separators of a graph. *International Journal of Foundations of Computer Science*, 11(3):397–403, 2000.

[13] Anne Berry, Jean-Paul Bordat, Pinar Heggernes, Geneviève Simonet, and Yngve Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. *Journal of Algorithms*, 58(1):33–66, 2006.

[14] Umberto Bertelè and Francesco Brioschi. On non-serial dynamic programming. *Journal of Combinatorial Theory, Series A*, 14(2):137–148, 1973.

[15] Hans L. Bodlaender. Discovering treewidth. In Peter Vojtáš, Mária Bieliková, Bernadette Charron-Bost, and Ondrej Sýkora, editors, *Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science*, volume 3381 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.

[16] Hans L. Bodlaender and Fedor V. Fomin. Tree decompositions with small cost. *Discrete Applied Mathematics*, 145(2):143–154, 2005.

[17] Hans L. Bodlaender, Pinar Heggernes, and Yngve Villanger. Faster parameterized algorithms for minimum fill-in. *Algorithmica*, 61(4):817–838, 2011.

[18] Hans L. Bodlaender and Arie M.C.A. Koster. Safe separators for treewidth. *Discrete Mathematics*, 306(3):337–350, 2006.

[19] Hans L. Bodlaender and Arie M.C.A. Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208(3):259–275, 2010.

[20] Hans L Bodlaender, Leizhen Cai, Jianer Chen, Michael R. Fellows, Jan Arne Telle, and Dániel Marx. Open problems in parameterized and exact computation – IWPEC 2006. Technical report UU-CS-2006-052, Department of Information and Computing Sciences, Utrecht University, 2006.

[21] Magnus Bordewich, Katharina T. Huber, and Charles Semple. Identifying phylogenetic trees. *Discrete Mathematics*, 300(1-3):30–43, 2005.

[22] Vincent Bouchitté and Ioan Todinca. Listing all potential maximal cliques of a graph. *Theoretical Computer Science*, 276(1-2):17–32, 2002.

[23] Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM Journal on Computing*, 31(1):212–232, 2001.

[24] Daniel R. Brooks, Esra Erdem, Selim T. Erdoğan, James W. Minett, and Don Ringe. Inferring phylogenetic trees using answer set programming. *Journal of Automated Reasoning*, 39(4):471–511, 2007.

[25] Peter Buneman. A characterisation of rigid circuit graphs. *Discrete Mathematics*, 9(3):205–212, 1974.

[26] Michael Coulombe, Kristian Stevens, and Dan Gusfield. Construction, enumeration, and optimization of perfect phylogenies on multi-state data. In *Proceedings of the 5th IEEE International Conference on Computational Advances in Bio and Medical Sciences*, pages 1–6. IEEE Computer Society, 2015.

[27] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[28] Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MaxSAT. In Matti Järvisalo and Allen Van Gelder, editors, *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013.

[29] William H. E. Day and David Sankoff. Computational complexity of inferring phylogenies by compatibility. *Systematic Zoology*, 35(2):224–229, 1986.

[30] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[31] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[32] Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. The first parameterized algorithms and computational experiments challenge. In Jiong Guo and Danny Hermelin, editors, *Proceedings of the 11th International Symposium on Parameterized and Exact Computation*, volume 63 of *LIPIcs*, 30:1–30:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[33] Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. In Daniel Lokshtanov and Naomi Nishimura, editors, *Proceedings of 12th International Symposium on Parameterized and Exact Computation*, volume 89 of *LIPIcs*, 30:1–30:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

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

[35] Feodor F. Dragan and Ekkehard Köhler. An approximation algorithm for the tree t-spanner problem on unweighted graphs via generalized chordal graphs. *Algorithmica*, 69(4):884–905, 2014.

[36] M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. The PACE 2019 parameterized algorithms and computational experiments challenge: The fourth iteration (invited paper). In Bart M. P. Jansen and Jan Arne Telle, editors, *Proceedings of the 14th International Symposium on Parameterized and Exact Computation*, volume 148 of *LIPIcs*, 25:1–25:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[37] Finnish Grid and Cloud Infrastructure, 2019. urn:nbn:fi:research-infras-2016072533.

[38] Johannes K. Fichte, Markus Hecher, Neha Lodha, and Stefan Szeider. An SMT approach to fractional hypertree width. In John Hooker, editor, *Proceedings of the 24nd International Conference on Principles and Practice of Constraint Programming*, volume 11008 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2018.

[39] Johannes K. Fichte, Markus Hecher, Stefan Woltran, and Markus Zisser. Weighted model counting on the GPU by exploiting small treewidth. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *Proceedings of the 26th Annual European Symposium on Algorithms*, volume 112 of *LIPIcs*, 28:1–28:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[40] Wolfgang Fischl, Georg Gottlob, Davide Mario Longo, and Reinhard Pichler. HyperBench: A benchmark and tool for hypergraphs and empirical findings. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 464–480, 2019.

[41] Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and fractional hypertree decompositions: Hard and easy cases. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 17–32, 2018.

[42] Fedor V. Fomin, Dieter Kratsch, Ioan Todinca, and Yngve Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM Journal on Computing*, 38(3):1058–1079, 2008.

[43] Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015.

[44] Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. *Combinatorica*, 32(3):289–308, 2012.

[45] Masanobu Furuse and Koichi Yamazaki. A revisit of the scheme for computing treewidth and minimum fill-in. *Theoretical Computer Science*, 531:66–76, 2014.

[46] Philippe Galinier, Michel Habib, and Christophe Paul. Chordal graphs and their clique graphs. In Manfred Nagl, editor, *Proceedings of the 21st International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 1017 of *Lecture Notes in Computer Science*, pages 358–371. Springer, 1995.

[47] A. Gewirtz. Graphs with maximal even girth. *Canadian Journal of Mathematics*, 21:915–934, 1969.

[48] Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. In Max Chickering and Joseph Halpern, editors, *Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence*, pages 201–208. AUAI Press, 2004.

[49] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.

[50] Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: NP-hardness and tractable variants. *Journal of the ACM*, 56(6):30:1–30:32, 2009.

[51] Georg Gottlob and Marko Samer. A backtracking-based algorithm for hypertree decomposition. *ACM Journal of Experimental Algorithmics*, 13:1.1:1–1.1:19, 2008.

[52] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1):4:1–4:20, 2014.

[53] Dan Gusfield. The multi-state perfect phylogeny problem with missing and removable data: Solutions via integer-programming and chordal graph theory. *Journal of Computational Biology*, 17(3):383–399, 2010.

[54] Dan Gusfield, Yelena Frid, and Dan Brown. Integer programming formulations and computations solving phylogenetic and population genetic problems with missing or genotypic data. In Guohui Lin, editor, *Proceedings of the 13th Annual International Conference on Computing and Combinatorics*, volume 4598 of *Lecture Notes in Computer Science*, pages 51–64. Springer, 2007.

[55] Rob Gysel. Minimal triangulation algorithms for perfect phylogeny problems. In Adrian-Horia Dediu, Carlos Martín-Vide, José Luis Sierra-Rodríguez, and Bianca Truthe, editors, *Proceedings of the 8th International Conference on Language and Automata Theory and Applications*, volume 8370 of *Lecture Notes in Computer Science*, pages 421–432. Springer, 2014.

[56] Rob Gysel and Dan Gusfield. Extensions and improvements to the chordal graph approach to the multistate perfect phylogeny problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(4):912–917, 2011.

[57] Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8(1–2):171–186, 1976.

[58] Masami Hasegawa, Hirohisa Kishino, and Taka-aki Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *Journal of Molecular Evolution*, 22(2):160–174, 1985.

[59] Eric P. Hoberg. Congruent and synchronic patterns in biogeography and speciation among seabirds, pinnipeds, and cestodes. *Journal of Parasitology*, 78(4):601–615, 1992.

[60] Richard R. Hudson. Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–338, 2002.

[61] IBM ILOG. CPLEX optimizer 12.7.1, 2017.

[62] Finn V. Jensen and Frank Jensen. Optimal junction trees. In Ramon Lopez de Mantaras and David Poole, editors, *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence*, pages 360–366. Morgan Kaufmann, 1994.

[63] Finn V. Jensen and Thomas D. Nielsen. *Bayesian networks and decision graphs*. Springer, 2007.

[64] David S. Johnson and Michael A. Trick, editors. *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1996. DIMACS/AMS.

[65] Sampath Kannan and Tandy Warnow. A fast algorithm for the computation and enumeration of perfect phylogenies. *SIAM Journal on Computing*, 26(6):1749–1763, 1997.

[66] Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst case and beyond. *ACM Transactions on Database Systems*, 41(4):22:1–22:45, 2016.

[67] Uffe Kjærulff. Triangulation of graphs — Algorithms giving small total state space. Research Report R-90-09, Department of Mathematics and Computer Science, Aalborg University, Denmark, 1990.

[68] Martin Kneser. Aufgabe 360. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 58, 1956.

[69] Tuukka Korhonen. Triangulator. https://github.com/Laakeri/triangulator-msc, 2020.

[70] Tuukka Korhonen, Jeremias Berg, and Matti Järvisalo. Enumerating potential maximal cliques via SAT and ASP. In Sarit Kraus, editor, *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 1116–1122. AAAI Press, 2019.

[71] Tuukka Korhonen, Jeremias Berg, and Matti Järvisalo. Solving graph problems via potential maximal cliques: An experimental evaluation of the Bouchitté–Todinca algorithm. *ACM Journal of Experimental Algorithmics*, 24(1):1.9:1–1.9:19, 2019.

[72] Tuukka Korhonen and Matti Järvisalo. Finding most compatible phylogenetic trees over multi-state characters. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*. AAAI Press, 2020.

[73] Arie M. C. A. Koster, Stan P. M. van Hoesel, and Antoon W. J. Kolen. Solving partial constraint satisfaction problems with tree decomposition. *Networks*, 40(3):170–180, 2002.

[74] Philipp Klaus Krause. Optimal register allocation in polynomial time. In Ranjit Jhala and Koen De Bosschere, editors, *Proceedings of the 22nd International Conference on Compiler Construction*, volume 7791 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2013.

[75] Philipp Klaus Krause, Lukas Larisch, and Felix Salfelder. The tree-width of C. *Discrete Applied Mathematics*, 278:136–152, 2020.

[76] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

[77] Chao Li and Maomi Ueno. An extended depth-first search algorithm for optimal triangulation of Bayesian networks. *International Journal of Approximate Reasoning*, 80:294–312, 2017.

[78] Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 613–631. IOS Press, 2009.

[79] Yang Li, Lloyd Allison, and Kevin Korb. Proving the NP-completeness of optimal moral graph triangulation. *CoRR*, abs/1903.02201, 2019. arXiv: 1903.02201.

[80] Daniel Lokshtanov. On the complexity of computing treelength. *Discrete Applied Mathematics*, 158(7):820–827, 2010.

[81] Joao Marques-Silva, Alexey Ignatiev, and Antonio Morgado. Horn maximum satisfiability: Reductions, algorithms and applications. In Eugénio C. Oliveira, João Gama, Zita A. Vale, and Henrique Lopes Cardoso, editors, *Proceedings of the 18th EPIA Conference on Artificial Intelligence*, volume 10423 of *Lecture Notes in Computer Science*, pages 681–694. Springer, 2017.

[82] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983.

[83] Ole J. Mengshoel. Understanding the scalability of Bayesian network inference using clique tree growth curves. *Artificial Intelligence*, 174(12-13):984–1006, 2010.

[84] James W. Minett and William S.-Y. Wang. On detecting borrowing: Distance-based and character-based approaches. *Diachronica*, 20(2):289–330, 2003.

[85] Miguel Miranda, Inês Lynce, and Vasco Manquinho. Inferring phylogenetic trees using pseudo-Boolean optimization. *AI Communications*, 27(3):229–243, 2014.

[86] Lukas Moll, Siamak Tazari, and Marc Thurley. Computing hypergraph width measures exactly. *Information Processing Letters*, 112(6):238–242, 2012.

[87] Kazuhide Nakata, Katsuki Fujisawa, Mituhiro Fukuda, Masakazu Kojima, and Kazuo Murota. Exploiting sparsity in semidefinite programming via matrix completion II: implementation and numerical results. *Mathematical Programming, Series B*, 95(2):303–327, 2003.

[88] Thorsten J. Ottosen and Jiří Vomlel. All roads lead to rome – New search methods for the optimal triangulation problem. *International Journal of Approximate Reasoning*, 53(9):1350–1366, 2012.

[89] Noam Ravid, Dori Medini, and Benny Kimelfeld. Ranked enumeration of minimal triangulations. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 74–88. ACM, 2019.

[90] Don Ringe, Tandy Warnow, and Ann Taylor. Indo-European and computational cladistics. *Transactions of the Philological Society*, 100(1):59–129, 2002.

[91] Neil Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of treewidth. *Journal of Algorithms*, 7(3):309–322, 1986.

[92] Donald J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In Ronald C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.

[93] Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.

[94] André Schidler and Stefan Szeider. Computing optimal hypertree decompositions. In Guy E. Blelloch and Irene Finocchi, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments*, pages 1–11. SIAM, 2020.

[95] Marco Scutari. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010.

[96] Charles Semple and Mike Steel. *Phylogenetics*. Oxford University Press, 2003.

[97] Kristian Stevens and Dan Gusfield. Reducing multi-state to binary perfect phylogeny with applications to missing, removable, inserted, and deleted data. In Vincent Moulton and Mona Singh, editors, *Proceedings of the 10th International Workshop on Algorithms in Bioinformatics*, volume 6293 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 2010.

[98] Hisao Tamaki. Computing treewidth via exact and heuristic lists of minimal separators. In Ilias Kotsireas, Panos M. Pardalos, Konstantinos E. Parsopoulos, Dimitris Souravlias, and Arsenis Tsokas, editors, *Proceedings of the Special Event on Analysis of Experimental Algorithms*, volume 11544 of *Lecture Notes in Computer Science*, pages 219–236. Springer, 2019.

[99] Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019.

[100] Robert E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985.

[101] Robert E. Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[102] Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

Appendix A Summary of Empirical Comparison

The following table lists the number of test instances in each problem and the numbers of solved instances, timeouts and memouts/runtime errors for each of the implementations. For Triangulator the memouts/runtime errors are all memouts. For other implementations we cannot reliably distinguish memouts from runtime errors.

Problem	Implementation	Solved	Timeouts	Memouts/RTEs	Instances
Treewidth	Triangulator	4826	358	257	5441
	PIDDT	5135	306	0	
	p17	5116	291	34	
	QuickBB	4201	1240	0	
Minimum fill-in	Triangulator	203	53	74	330
	MCCP	122	82	126	
	PIDDM	195	135	0	
GHTW	Triangulator	2544	405	121	3070
	FraSMT	1494	1575	1	
	BalSep	1012	1856	202	
	G-BIP	858	2203	9	
	L-BIP	901	2159	10	
FHTW	Triangulator	2425	523	122	3070
	FraSMT	2010	1044	16	
HTW	det-k-decomp	1455	1606	9	3070
TTS	Triangulator	15	7	2	24
	EDFS	15	8	1	
Perf. Phyl.	Triangulator	882	138	0	1020
	Minsep IP	787	233	0	
	Bin IP	573	97	350	
	PerftPhy	939	79	2	
Bin. Maxcomp	Triangulator	376	416	8	800
	Minsep IP	111	675	14	
	Bin IP	576	23	201	
MS Maxcomp	Triangulator	1152	693	0	1845
	Minsep IP	804	1036	5	
	Bin IP	472	617	756	
Treelength	Triangulator	184	10	36	230