Department of Mathematics & Computer Science
Algorithmics Research Group

Master-Thesis

Cluster Deletion on Unit Disk Graphs

Author:	Sebastian Ochs
Supervisors:	Prof. Dr. Christian Komusiewicz
	Dr. Frank Sommer
	Jaroslav Garvardt, M. Sc.

September 30, 2023

Eigenständigkeitserklärung

Hiermit versichere ich, Sebastian Ochs, dass ich die vorliegende Arbeit mit dem Titel ,,Cluster Deletion on Unit Disk Graphs" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Diese Arbeit wurde in dieser oder ähnlicher Form noch keiner anderen Hochschule vorgelegt und hat noch keiner anderen Prüfungsleistung gedient.

Marburg, den 30.09.2023

Abstract.

For an undirected graph G and a natural number k, the Cluster Deletion problem asks whether G can be transformed into a cluster graph by deleting at most k edges. The related Cluster Editing problem asks whether G can be transformed into a cluster graph by k edge modifications, that is, the addition or the deletion of an edge. The intersection graphs of unit diameter circles in the two-dimensional plane are called unit disk graphs. In this work, we show that Cluster Deletion and Cluster Editing are NP-hard even on planar unit disk graphs with maximum degree 4. Under the exponential time hypothesis (ETH), we show a running time lower bound of $2^{o(\sqrt{n+m})}$ for Cluster Deletion and Cluster Editing, also on planar unit disk graphs with maximum degree 4. In the second part of this work, we present an FPT-algorithm for Cluster Deletion parameterized by treewidth ω. Our algorithm has a running time of $O(3^{\omega} \cdot \omega \cdot n)$, provided that a nice tree decomposition of width ω is given. We conclude that on unit disk graphs of constant maximum degree as well as on planar graphs Cluster Deletion can be solved in $2^{O(\sqrt{n})} \cdot n^{1.5}$ time, which matches our running time lower bound up to a polynomial factor.

Contents

1 Introduction

Clustering is the process of splitting a set of objects into groups so that the objects within a group are similar and objects from different groups are dissimilar. As a form of exploratory data analysis, clustering is a central technique in unsupervised machine learning with many fields of application, including pattern recognition, data compression, bioinformatics and business intelligence [30]. Naturally, there are many ways of formulating clustering problems. Several of them are defined on graphs, where two objects, represented by vertices, are similar if there is an edge between them. Two well-studied graph clustering problems are CLUSTER EDITING and CLUSTER DELETION. In both problems, the goal is to transform a graph into a cluster graph, where any two vertices share an edge if and only if they are in the same group. The decision version of CLUSTER DELETION asks whether it is possible to transform a graph G into a cluster graph by deleting k edges. The decision version of CLUSTER EDITING asks whether it is possible to transform a graph G into a cluster graph by k edge modifications, that is, additions or deletions. The optimization version of any of the two problems asks for the smallest number k so that the problem can be solved. In the general case, CLUSTER EDITING and CLUSTER DELETION are known to be NP-hard [54, 64]. For many different graph classes, the complexity analysis of both problems has been settled, as efficient algorithms and running time lower bounds are known.

In this work we study the problem of CLUSTER DELETION on unit disk graphs, a subclass of simple undirected graphs. Unit disk graphs are the intersection graphs of unit diameter circles in the two-dimensional plane. For a set of points $P \subset \mathbb{R}^2$, the corresponding unit disk graph is created the following way: For each point in P, we create a vertex. We then connect any two vertices by an edge if and only if the distance between their associated points is at most 1. Any graph that can be created in this way is a unit disk graph. The set of points P is called a unit disk representation of the graph. Different sets of equal size can have the same corresponding unit disk graph. This definition gives rise to the hope that the geometrical properties of unit disk graphs can be used to create efficient, perhaps even polynomial time algorithms. To the best of our knowledge, neither CLUSTER DELETION nor CLUSTER EDITING has yet been explored on unit disk graphs.

1.1 Related Work

The NP-hardness of Cluster Editing has been shown even on graphs with maximum degree 5 [31]. Cluster Deletion is NP-hard even on graphs with maximum degree 4 and solvable in $O(n^{1.5} \log^2(n))$ time on graphs with maximum degree 3 or less [49]. Note that we use n to denote the number of vertices in a graph and m to denote the number of edges in a graph. Cluster Editing is also known under the name Correlation Clustering [6], whereas Cluster Deletion is sometimes called Maximum Edge Clique Partition [29]. A cluster graph is eqivalently defined as a P_3-free graph, that is, a graph that contains no path on three vertices (P_3) as an induced subgraph.

Cluster Editing and Cluster Deletion are fixed-parameter tractable for the number k of edge modifications required to transform the graph G into a cluster graph [18]: Both problems can be solved by recursively searching a P_3 and considering the possible edge modifications to destroy it. For Cluster Deletion, this results in a running time of $O(2^k \cdot n^3)$, as for each P_3, two possible edge deletions need to be considered. For Cluster Editing, this results in a running time of $O(3^k \cdot n^3)$, as one must also consider adding an edge to turn the P_3 into a triangle [18]. There have been multiple improvements to this approach [40, 39, 8, 27]. They make use of a wide range of techniques, such as solving connected components independently, merging vertices which are to end up in the same cluster and calculating lower bounds to terminate branches early. For Cluster Editing, the current best FPT-algorithm has a running time of $O(1.62^k + n + m)$ [10], whereas, for Cluster Deletion a running time of $O^*(1.415^k)$ has been achieved [7].

One reason why the parameter k of edge modifications has attracted attention is that there exits a $2k$-vertex kernel for both Cluster Editing and Cluster Deletion [21][19]: It is possible in $O(n+m)$ time to transform any instance (G, k) of Cluster Editing into an equivalent instance (G', k') with $k' \leq k$ so that G' has at most $2k'$ vertices. The same algorithm also produces a $2k$-vertex kernel for Cluster Deletion and the related problem of Strong Triadic Closure [20].

Under the exponential time hypothesis (ETH), both Cluster Editing and Cluster Deletion cannot be solved in $2^{o(n+m)}$ time [49]. Therefore, any future improvements can only be expected to reduce the base of the exponential function of the running times mentioned above. In search for subexponential or even polynomial running times for Cluster Editing and Cluster Deletion, one has to restrict the properties of the input graph to a subclass. Cluster Deletion can be solved in polynomial time on graphs where no edge is contained in three different P_3s [27]. Cluster Deletion can also be solved in polynomial time on cographs, that is, graphs which do not contain a P_4 [35]. In contrast, Cluster Deletion is NP-hard even on P_5-free chordal graphs [15], aswell as on planar graphs [38, 48]. Cluster Deletion can be solved in polynomial time on (unit) interval graphs [15, 51]. The class of (unit) interval graphs is defined as the intersection graphs of (unit) intervals. They can, therefore, be considered the one-dimensional pendant to (unit) disk graphs.

One of the central questions adressed in this work is the complexity of Cluster Deletion on the class of unit disk graphs. There are several induced subgraphs which cannot exist in unit disk graphs [5]. However, there are up to $2^{\frac{n}{2}}$ maximal cliques in unit disk graphs [69, 41] compared to $3^{\frac{n}{3}}$ maximal cliques in general graphs. Although the upper bound is lower for unit disk graphs, the number of maximal cliques still grows exponentially with the number of vertices. Recognizing whether a graph is a unit disk graph, that is, whether it has a unit disk representation, is NP-hard [17]. The best-known problem that is NP-hard in the general case but solvable in polynomial time on unit disk graphs is Clique. It is possible in $O(n^{3.5} \cdot \log(n))$ time to find the biggest clique in a given unit disk graph [16]. This is true even if no unit disk representation is given [63]. Independent Set, however, is NP-hard on unit disk graphs [22]. The NP-hardness of many other problems on unit disk graphs has also been proven, including: Vertex Cover [22], Hamiltonian Cycle [45], Steiner Tree [36] and 3-Coloring [22].

Although most well-known graph problems that are NP-hard in the general case are also NP-hard on unit disk graphs, there is some hope for finding better running times: Many problems that, under the ETH, have a running time lower bound of $2^{o(n)}$ in the general case, have subexponential or FPT-algorithms on unit disk graphs. Independent Set can be solved in $O(2^{\sqrt{n}})$ and in $O(2^{\sqrt{k}} + n)$ time on unit disk graphs [1]. Moreover, it is possible to decide whether a given unit disk graph contains a path/cycle induced by k vertices in $2^{O(\sqrt{k} \cdot log(k))} \cdot n^{O(1)}$ time [32]. Under the ETH, this running time is tight up to the logarithmic factor in the exponent [32]. Also the k-Coloring problem can be solved in $2^{O(\sqrt{n \cdot k} \cdot log(n))}$ time on unit disk graphs. Again this running time is ETH-tight up to the logarithmic factor in the exponent [25]. Clique Vertex Cover and Clique Edge Cover can be solved in $O(n^{6k+2})$ time and $O(n^{6k+3})$ time on unit disk graphs [42]. They ask whether it is possible to cover the vertices/edges of a graph by selecting k cliques. Both problems are, therefore, related to clustering problems.

With a recently presented framework it is possible to design subexponential algorithms and matching ETH lower bounds for problems on intersection graphs of similarly-sized geometric objects [28]: There exist algorithms with running times of $2^{O(\sqrt{n})}$ and, under the ETH, matching lower bounds for many problems such as Independent Set, r-Dominating Set for constant r, Steiner Tree, Hamilton Cycle/Path on unit disk graphs [28].

Another recently published framework makes use of the fact that disk graphs of bounded clique size have bounded treewidth [58]. It can be used to design subexponential FPT-algorithms for problems on disk graphs: For some constant $\alpha < 1$ there exist algorithms with running times of $2^{O(k^{\alpha})}n^{O(1)}$ for Triangle Hitting, Feedback Vertex Set and Odd Cycle Transversal on disk graphs [58].

One general observation for problems on unit disk graphs is that many running time lower and upper bounds are of the form $2^{O(\sqrt{n})}$ or similar to it. A graph class where the same observation can be made is the class of planar graphs. This has been called the *square root phenomenon* of planar graphs [62]. It can be attributed to the fact that any planar graph has a separator consisting of $O(\sqrt{n})$ vertices [56]. A stronger result is that any planar graph has treewidth $\omega < 3.182\sqrt{n}$ [34].

The graph parameter treewidth ω is commonly described as a measure of how tree-like a graph is. A tree has a treewidth of 1, whereas a clique consisting of c vertices has a treewidth of $c-1$. Since unit disk graphs can have arbitrarily large cliques, they can also have arbitrarily large treewidth. If we do, however, restrict the maximum degree Δ of a unit disk graph to a constant, we get the following bound on its treewidth: $\omega \in O(\Delta^3\sqrt{n})$ [33]. There are other graph classes that have bounded treewidth as well [13, 65]. For this reason, much research has been devoted to the design of FPT-algorithms for the parameter treewidth [11, 67, 57, 26]. The problems of Minimum Clique Partition and Maximum μ-Clique Packing can be solved in $O(2^{\omega} \cdot \text{poly}(\omega) \cdot n)$ if a tree decomposition of width ω is given [68]. This is notable as both problems can be considered graph clustering problems. It is known that Cluster Deletion is fixed-parameter tractable for the parameter treewidth [50]. To the best of our knowledge, no FPT-algorithm for Cluster Deletion parameterized by treewidth has yet been presented.

1.2 Our Results

In Chapter 3 we present a polynomial-time reduction of 3-Sat to Cluster Deletion and Cluster Editing on unit disk graphs. More precisely, we show that both Cluster Deletion and Cluster Editing remain NP-hard, even when restricted to planar unit disk graphs with maximum degree 4. For Cluster Editing, this is the first proof of NP-hardness on graphs of maximum degree 4, as previously NP-hardness was known only on graphs with maximum degree 5 [31]. Assuming the exponential time hypothesis (ETH), we show that both Cluster Deletion and Cluster Editing cannot be solved in $2^{o(\sqrt{n+m})}$ time on planar unit disk graphs with maximum degree 4.

In Chapter 4 we present an FPT-algorithm for Cluster Deletion parameterized by treewidth ω that has a running time of $O(3^{\omega} \cdot \omega \cdot n)$ if a nice tree decomposition of width ω is given and $2^{O(\omega)} \cdot \omega \cdot n$ otherwise. This answers the question of whether such an algorithm admits a single exponential running time, recently mentioned by Italiano et al. [46]. Furthermore, we conclude that Cluster Deletion can be solved in $2^{O(\sqrt{n})} \cdot n^{1.5}$ time on planar graphs and on unit disk graphs of constant maximum degree, as both graph classes are subject to the bound $\omega \in O(\sqrt{n})$ [34][33]. We observe that this running time matches the lower bound stated in Chapter 3 up to a polynomial factor.

2 Preliminaries

In this chapter we will explain the terminology and specify the notations used throughout this work. We first give a brief introduction to the fields of boolean algebra and computational complexity theory. We then explain the notations used to describe graphs and state the definitions of the different graph classes considered in this work. Subsequently, we define all graph problems considered in terms of their inputs and outputs. Finally, we state some simple observations about Cluster Deletion which will help to simplify further statements.

2.1 Boolean Algebra

In the following, we provide a brief introduction to boolean algebra, a fundamental cornerstone of computer science. Boolean algebra is based on the values `true` and `false`. With the operator NOT, denoted by an overline, values can be negated: $\overline{\texttt{true}} = \texttt{false}$. The operator AND, denoted by $\wedge$, takes in two values and evaluates to `true` if and only if both of them are `true`. The operator OR, denoted by $\vee$, takes in two values and evaluates to `true` if at least one of them is `true`. Operators can be combined with boolean variables to express logical propositions, called *boolean formulas*. Let $\mathcal{X}$ be a set of boolean variables. An *assignment* $\alpha : \mathcal{X} \to \{\texttt{true}, \texttt{false}\}$ maps every variable of $\mathcal{X}$ to a boolean value. A boolean formula Φ over the variable set $\mathcal{X}$ is a function that maps each possible assignment of $\mathcal{X}$ to a boolean value. The formula Φ is *satisfiable* if and only if there exists an assignment for its variables so that Φ evaluates to `true`. The occurrences of the variables in the formula itself, together with a possible negation, are referred to as *literals* $L = \{\ell_1, ..., \ell_{|L|}\}$ where $\ell_i \in \{x \mid x \in \mathcal{X}\} \cup \{\overline{x} \mid x \in \mathcal{X}\}$ for $i \in \mathbb{N}$, $i \leq |L|$. Note that L is a multiset, as literals can appear multiple times. A *clause* consists of literals which are joined together by operators. For a multiset of literals $L = \{\ell_1, \ell_2 ...\}$, a clause of the form $(\ell_1 \wedge \ell_2 \wedge ...)$ is called a *conjunction*, whereas a clause of the form $(\ell_1 \vee \ell_2 \vee ...)$ is called a *disjunction*. A formula Φ is in *conjunctive normal form (CNF)* if it is a conjunction of disjunction-clauses. Furthermore, a formula Φ is in *3-CNF* if it is in CNF and every disjunction contains at most three literals. The 3-Sat problem is defined in the following way.

3-Sat

Input:	A boolean formula Φ in 3-CNF
Question:	Is Φ satisfiable ?

Without loss of generality, we assume that for any instance of 3-Sat, in the formula Φ all clauses consist of exactly 3 literals. We also assume that each variable appears at least twice, as otherwise its assignment would be trivial.

Let Φ be a formula in 3-CNF with the variable set $\mathcal{X}$ and the clause set $\mathcal{C}$. The *associated graph* of Φ is created in the following way: We create a vertex for every variable in $\mathcal{X}$ and for every clause in $\mathcal{C}$. We then create an edge for every literal in Φ from the vertex of its clause to the vertex of its variable. The problem of Planar 3-Sat is defined similarly to 3-Sat with the only difference being that the associated graph of the input formula Φ is planar. Both 3-Sat and Planar 3-Sat are NP-complete [23, 55].

2.2 Computational Complexity Theory

The field of computational complexity theory aims to answer questions about the requirements necessary to solve computational problems in terms of space and running time. In the following, we briefly go over some of the key concepts in the field. A detailed introduction can be found in several textbooks [4, 37].

Running Time: The number of steps it takes an algorithm to terminate in the worst-case is called its *running time*. When analyzing the running time of an algorithm, the goal is to bound its running time by a function. The variables of this running time function can depend on both the input and the output of the algorithm.

Decision Problem: For a *finite alphabet* Σ a *language* L is a subset of Σ^*. The question of whether a *word* $x \in \Sigma^*$ is part of the language $L \subseteq \Sigma^*$ is the *decision problem* $P_L : \Sigma^* \to \{\texttt{true}, \texttt{false}\}$. We have:

$P_L(x) = \texttt{true} \quad \Leftrightarrow \quad x \in L \quad \Leftrightarrow \quad x$ is a *yes-instance* of P_L.

Reduction: Let L_1 and L_2 be two languages. A *reduction* from P_{L1} to P_{L2} $r : \Sigma^* \rightarrow \Sigma^*$ is a computable function so that $x \in L_1 \Leftrightarrow r(x) \in L_2$. We call r a *polynomial time reduction* from P_{L1} to P_{L2} if and only if there exists an algorithm that, for each $x \in \Sigma^*$, computes the result $r(x)$ in a running time polynomial in $|x|$. In other words: For an instance p_1 of a decision problem P_1 a reduction constructs an equivalent instance p_2 of another problem P_2. We could now solve the instance p_1 by solving p_2. This way, we can show that solving the problem P_1 takes at most the time required to perform the reduction and subsequently solve the problem P_2.

P vs. NP: In the field of complexity theory desicion problems are divided into different categories called *complexity classes*. The class P is made up of problems that can be solved in polynomial time by a deterministic turing machine. The class NP is made up of problems that can be solved in polynomial time by a non-deterministic turing machine and verified in polynomial time by a deterministic turing machine. We have P $\subseteq$ NP. The assumption that there are problems that cannot be solved in polynomial time by a deterministic turing machine, namely P $\neq$ NP, is widely believed to be true. A problem is *NP-hard* if there exists a polynomial time reduction from any problem in NP to it. A problem is *NP-complete* if it is both NP-hard and in NP. The complexity class of a problem tells us whether or not there exists an algorithm for it with a running time bounded by a polynomial function. It does, however, not tell us anything about the specific form of this function.

ETH: A stronger assumption, which implies P $\neq$ NP, is the *exponential time hypothesis (ETH)*. It states that for any integer $k \geq 3$, there exists a number $s_k > 0$ so that k-SAT cannot be solved in $2^{(s_k - \epsilon)n}$ time, where $\epsilon > 0$ [43]. Under the ETH it is possible to give running time lower bounds [44]: 3-SAT cannot be solved in $2^{o(|\mathcal{X}|+|\mathcal{C}|)}$ time, unless the ETH is false. Note that $\mathcal{X}$ and $\mathcal{C}$ are the variables and clauses of the given formula.

Further remarks: We assume that, in our computational model, basic arithmetic operations, such as adding or multiplying two integers, as well as querying a table for a single value, are possible in constant time. This is particularly relevant when analyzing the running time of dynamic algorithms on tree decompositions [14], as we do in Chapter 4.

2.3 Graphs

A *graph* $G = (V, E)$ consists of a set of *vertices* V and a set of *edges* $E \subseteq \{\{u, v\} : u, v \in V\}$. The sizes of those sets are denoted with $n = |V|$ and $m = |E|$. We say two vertices $u, v \in V$ are *adjacent* if $\{u, v\} \in E$. The edge $\{u, v\}$ is *incident* with the vertices u and v. For a vertex v we denote its *neighborhood* by $N(v) = \{u \in V : \{u, v\} \in E\}$. We refer to the vertices in $N(v)$ as the *neighbors* of v. The number of neighbors of a vertex v is denoted by $\text{degree}_G(v) = |N(v)|$. Note that we omit the subscript when the graph is clear from the context. The *maximum degree* of a graph G is defined as $\Delta = \max_{v \in V} \text{degree}(v)$.

Let $G = (V, E)$ be a graph. The graph $G' = (V', E')$ is called an *induced subgraph* of G if $V' \subseteq V$ and $E' = \{\{u, v\} \in E : u, v \in V'\}$. We also refer to G' as the graph *induced by* V'. For a vertex set $S \subseteq V$ we denote with $G - S$ the graph induced by the vertex set $V \backslash S$.

Let $G = (V, E)$ be a graph. A sequence of distinct vertices $\sigma = (v_1, ..., v_\ell)$ is a *path* of length ℓ if $\{v_i, v_{i+1}\} \in E$ for all $i \in [1, \ell - 1]$. It is also denoted as a P_ℓ or as a (v_1, v_ℓ)-path since it starts at v_1 and ends at v_ℓ. A *connected component* $C = (V', E')$ is a maximal induced subgraph of G so that for every pair of vertices $u, v \in V'$ there exists a (u, v)-path in G. Every vertex of a graph belongs to exactly one connected component. A graph is *connected* if it has only one connected component. All graphs considered in this work are *simple* and *undirected*, meaning edges have no weight and no direction.

Basic Graph Classes

A graph $G = (V, E)$ is a ...

... *triangle*, if we have $|V| = 3$ and $|E| = 3$.
... *diamond*, if we have $|V| = 4$ and $|E| = 5$.
... *(cordless-) cycle*, if it is connected and each vertex v has $\text{degree}(v) = 2$.
... *tree*, if it is connected and contains no cycle as an induced subgraph.
... *clique* of size $s \in \mathbb{N}$, if $|V| = s$ and $|E| = \binom{s}{2}$.
... *cluster graph*, if every connected component of G is a clique.

For a number $s \in \mathbb{N}$ the graph $G = (V, E)$ is called an *s-star* if it has a vertex $v_c \in V$ so that $\text{degree}(v_c) = s = |V| - 1 = |E|$. The vertex v_c is called the *center* of the s-star.

Planar Graphs

A graph is *planar* if it can be drawn in the two-dimensional plane in such a way that no two edges cross each other. More formally, vertices are represented by distinct points and edges are represented by non-intersecting curves between the points of their vertices.

Unit Disk Graphs

Let $P \subset \{(x, y) : x, y \in \mathbb{R}\}$ be a set of points in the two-dimensional plane. The *unit disk graph* corresponding to P is created the following way: For each point we create a vertex. We then connect each pair of vertices by an edge if and only if their points have a euclidean distance of at most 1. A graph $G = (V, E)$ is a *unit disk graph* if it can be created this way. In other words: G is a unit disk graph if there exists a set of points P so that G is equivalent to the unit disk graph corresponding to P. We call P a *unit disk representation* of G.

Tree Decompositions

A *tree decomposision* of a graph G is a graph $D = (I, \Lambda)$ with a set of nodes I that are connected by the edges Λ to form a tree. Each node $i \in I$ is associated with a vertex set $V_i \subseteq V$ that is called its *bag*, so that:

1. each vertex is part of at least one bag,
2. for each vertex v, the nodes that contain v in its bag induce a tree in D,
3. for each edge, both of its vertices are part of at least one bag.

A *nice tree decomposition* of a graph G is a tree decomposition of G that is a binary tree with the root node r so that the bag of r is empty, $V_r = \emptyset$, and furthermore, each node is of exactly one type:

1. a *leaf node* i has no child nodes and its bag is empty: $V_i = \emptyset$,
2. an *introduce node* i has one child node j so that $V_i = V_j \cup \{v\}$,
3. a *forget node* i has one child node j so that $V_i = V_j \backslash \{v\}$,
4. a *join node* i has two child nodes j and ℓ so that $V_i = V_j = V_\ell$.

Let G be a graph and let $D = (I, \Lambda)$ be a nice tree decomposition of G. For a node $i \in I$ we denote with G_i the graph induced by the vertices in V_i and all vertices in bags of nodes that appear below i in D. Note that we also refer to i as a node $i \in D$.

The *width* ω of a (nice) tree decomposition D is defined as the maximum number of vertices in a bag minus one: $\omega = \max_{i \in D} |V_i| - 1$. The *treewidth* of a graph G is the smallest possible width ω of any tree decomposition D of G. Calculating the treewidth of a graph is NP-hard [3]. For a graph G and any fixed ω it is possible in $2^{O(\omega^3)} \cdot n$ time to compute a tree decomposition of width ω or to determine that the treewidth of G is greater than ω [12]. There is also a recent algorithm which achieves the same result in $2^{O(\omega^2)} \cdot n^4$ time [53]. As both running times contain a double exponential factor, to compute tree decompositions one often uses approximations with smaller running times. For a graph G and any fixed ω it is possible in $2^{O(\omega)} \cdot n$ time to compute a tree decomposition of width at most $2\omega + 1$ or to determine that the treewidth of G is greater than ω [52]. One can assume that any tree decomposition computed or given has at most n nodes. Otherwise it would be redundant in the sense that it would contain a node whose bag is a subset of another bag [2]. When given a tree decomposition of width ω, it is possible in $O(\omega^2 \cdot n)$ time to create a nice tree decomposition of width ω that has at most $O(\omega \cdot n)$ nodes [2].

The treewidth is commonly described as a measure of how treelike a graph is. A tree has $\omega = 1$ and a clique has $\omega = n - 1$. A graph with treewidth ω contains no clique of size $\omega + 2$.

2.4 Graph Problems

In the following, the graph problems considered in this work are formally defined. The two main problems adressed are Cluster Editing and Cluster Deletion. They are closely related, as their definitions show:

Cluster Deletion

Input:	A Graph $G = (V, E)$ and an integer k.
Question:	Can G be transformed into a cluster graph by deleting at most k edges ?

Cluster Editing

Input:	A Graph $G = (V, E)$ and an integer k.
Question:	Can G be transformed into a cluster graph by at most k edge modifications, that is, additions and deletions ?

Let G be a graph. The minimum number k for which (G, k) is a yes-instance of Cluster Deletion is greater than or equal to the minimum number k' for which (G, k') is a yes-instance of Cluster Editing.

The Clique Partition problem can also be considered a graph clustering problem. Similarly to Cluster Deletion, the goal is to group the vertex set of G into cliques. Clique Partition seeks to minimize the number of cliques, whereas Cluster Deletion seeks to maximize the number of edges within the cliques.

Clique Partition

Input:	A Graph $G = (V, E)$ and an integer k.
Question:	Can V be divided into k disjoint subsets $V_1, ..., V_k \subseteq V$ so that each subset induces a clique ?

μ-Clique Packing

Input:	A Graph $G = (V, E)$ and an integer k.
Question:	Is it possible to create k disjoint subsets $V_1, ..., V_k \subseteq V$ so that each subset induces a clique of size μ?

Note that these are the decision versions of the problems. They ask whether there exists a solution of a given size k. The optimization versions ask for the smallest/biggest k for which there exists a solution of size k.

2.5 Simple Observations for Cluster Deletion

In the following, we list some known reduction rules for instances of Cluster Deletion. We will not prove them, as they are straightforward and can be considered common knowledge. Additionally, we state two lower bounds on the number of edge deletions required to turn certain induced subgraphs into cluster graphs. These are also stated without a proof, but will help to simplify further statements.

Reduction Rule 1 *Let G be a graph with multiple connected components. We can solve* Cluster Deletion *independently on each connected component. The solution size k of* Cluster Deletion *on G is the sum of the solution sizes for all connected components.*

Reduction Rule 2 *For a graph $G = (V, E)$ let $\{u, v\}, \{v, w\} \in E$ be edges. Further let* $\text{degree}(u) = 1$ *and* $\text{degree}(v) = 2$*. We can delete the edge $\{v, w\}$ from G and reduce k by 1.*

Reduction Rule 3 *For a graph $G = (V, E)$ let $\{v_1, v_2, v_3\}$ be a triangle so that* $\text{degree}(v_1) + \text{degree}(v_2) + \text{degree}(v_3) = 7$*. Let $u \in V$ be the only neighbor of this triangle so that $\{v_1, u\} \in E$. We can delete the edge $\{v_1, u\}$ and reduce k by 1.*

Reduction Rule 4 *For a number $s \in \mathbb{N}$ let $G = (V, E)$ be an s-star. We can delete all edges except one and reduce k by $s - 1$.*

Reduction Rule 5 *For a graph $G = (V, E)$ let $\{v_1, v_2, v_3, v_4\} \in V$ be vertices that induce a P_4 with the edges $\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\} \in E$. Let $N(v_1) \cap N(v_4) = \emptyset$,* $\text{degree}(v_2) = 2$ *and* $\text{degree}(v_3) = 2$*. We can remove the vertices v_2 and v_3 as well as the edges incident with them, add the edge $\{v_1, v_4\}$ and reduce k by 1.*

Lower Bound 1 *For a number $\ell \in \mathbb{N}$, let G be a graph that contains a P_ℓ as an induced subgraph. In any optimal solution of* Cluster Deletion *on G, at least $\lfloor \frac{\ell-1}{2} \rfloor$ edges of this P_ℓ are deleted.*

Lower Bound 2 *For a number $s \in \mathbb{N}$, let G be a graph that contains an s-star G_s as an induced subgraph. In any optimal solution of* Cluster Deletion *on G, at least $s - 1$ edges of G_s are deleted.*

3 Running Time Lower Bound

In this chapter we will study the complexity of Cluster Deletion on unit disk graphs.

Theorem 1 Cluster Deletion *is NP-hard even on planar unit disk graphs with maximum degree* 4*.*

To prove Theorem 1, we present a polynomial-time reduction of 3-SAT to Cluster Deletion on unit disk graphs. In Section 3.1 we define two types of subgraphs and show how they help in expressing boolean formulas as equivalent instances of Cluster Deletion. Section 3.2 contains the formal description of our reduction. In Section 3.2.1 we define a set of gadgets by giving unit disk representations for them. In Section 3.2.2 we explain how the gadgets can be combined to construct a unit disk graph G from a given instance of 3-SAT. We also calculate the number of edge deletions needed to turn G into a cluster graph. In Section 3.2.3 we prove that the constructed instance of Cluster Deletion is equivalent to the given instance of 3-SAT. As our reduction takes only polynomial time, we obtain the NP-hardness of Cluster Deletion on unit disk graphs. The structural properties of the constructed unit disk graph imply the constraints in Theorem 1.

Theorem 2 Cluster Deletion *cannot be solved in* $2^{o(\sqrt{n+m})}$ *time on planar unit disk graphs with maximum degree* 4*, unless the ETH fails.*

In Section 3.2.4 we prove Theorem 2 by bounding the size of the graph constructed in our reduction.

Overview of Reduction

Most reductions from 3-SAT to graph problems have a common pattern: Based on a given formula Φ one constructs a graph consisting of so called gadgets for both the variables and clauses and then connects them according to Φ. A fundamental challenge when following this pattern to construct a unit disk graph is that we cannot create edges arbitrarily. The unit disk graph constructed in our reduction therefore has a clear layout, which is exemplified in Figure 1.

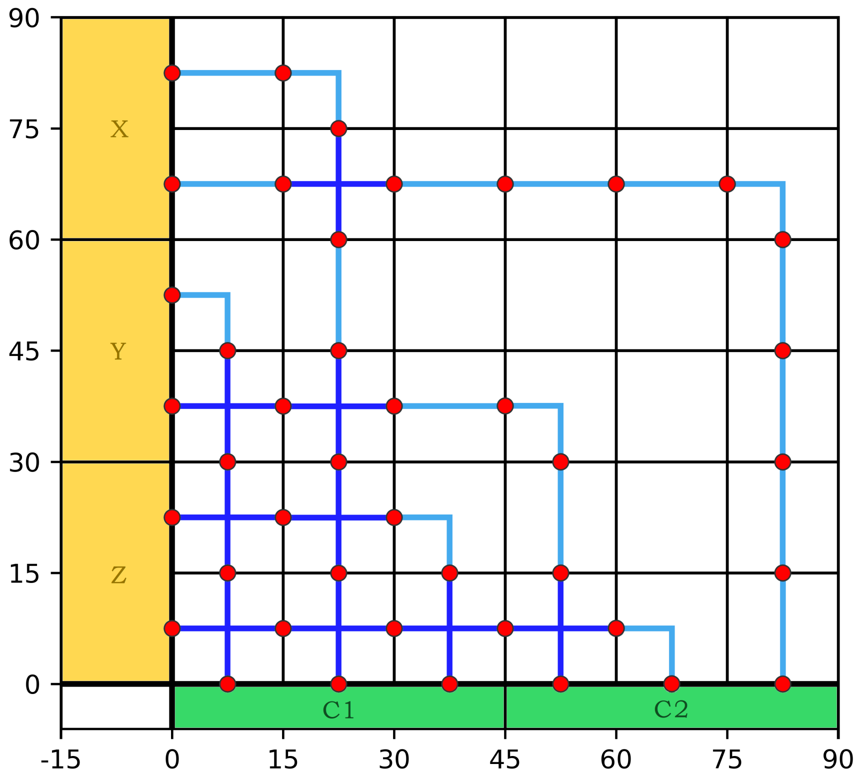


Figure 1: Schematic of a constructed unit disk graph with 3 Variable Gadgets (yellow), 2 Clause Gadgets (green), Cable Gadgets (light blue), Crossing Gadgets (dark blue) and Interface Vertices (red).

The two-dimensional space is divided into a grid so that we can distinguish between tiles of size 15 × 15 individually. The clause gadgets are lined up horizontally, just below the X-axis. The size of a clause gadget is fixed; it takes up three tiles, one for each literal. The variable gadgets are lined up vertically, just left of the Y-axis. The size of a variable gadget varies; it takes up one tile for each occurence of its variable in the formula Φ. The structure of the formula Φ, more specifically its associated graph, is modeled by cable gadgets. Each cable gadget takes up a single tile. For each literal we form a path consisting of cable gadgets from the corresponding clause gadget to the corresponding variable gadget. In tiles where two cable gadgets would cross each other we place a crossing gadget instead. This is necessary because due to the nature of unit disk graphs placing two cable gadgets on top of each other would induce additional edges.

Our crossing gadget is inspired by the crossing gadget used in a reduction of 3-SAT to Planar 3-SAT to show its NP-hardness [55]. There are many more variants of Planar 3-SAT, a lot of them are NP-hard as well [66]. To show the NP-hardness of Cluster Deletion on unit disk graphs a reduction from any of these variants would suffice. This would not require a crossing gadget, as by definition the associated graph of a formula in planar 3-CNF is planar. The reason we reduce from 3-SAT is that its running time lower bound, assuming the ETH, is stronger. In fact, the ETH itself implies that 3-SAT cannot be solved in $2^{o(|\mathcal{C}|)}$ time, where $\mathcal{C}$ is the number of clauses in the given 3-CNF formula [43].

3.1 Special Subgraphs

In this section we introduce two specific graph structures and calculate the number of edge deletions it takes to transform them into cluster graphs. We then show how they will help in our endeavor of expressing boolean formulas as instances of Cluster Deletion.

3.1.1 Triangle-Ring

The first structure is called a triangle-ring. As seen in Figure 2, it consists of an even number of triangles which form a ring so that any two neighboring triangles share a vertex.

Definition 1 (Triangle-Ring) *For $s \in \mathbb{N}$ with $s \bmod 2 = 0$ and $s \geq 4$, the graph $G = (V, E)$ is called a* triangle-ring *of size s if it can be constructed in the following way:*
1. *Construct a cordless cycle consisting of s vertices and s edges.*
2. *For each edge $\{u_1, u_2\}$ of this cycle add a vertex v with $N(v) = \{u_1, u_2\}$.*

When talking about a triangle-ring, the vertices of the cordless cycle are referred to as the *inner vertices*. The other vertices of a triangle-ring are referred to as the *outer vertices*. Note that each inner vertex has degree 4 and each outer vertex has degree 2. Further note that the size s is defined to be an even number, as all triangle-rings used in the reduction have even size. Finally note that a triangle-ring of size s has exactly $2 \cdot s$ vertices and $3 \cdot s$ edges. In the context of Cluster Deletion, the term *deleting the triangles alternatingly* refers to the deletion of the edges of every second triangle in the ring.

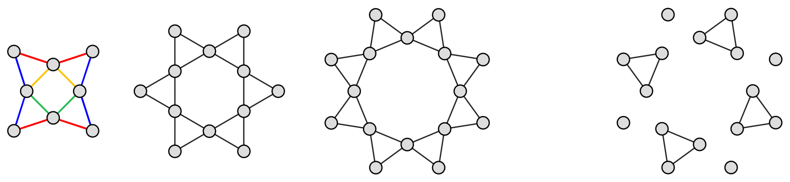
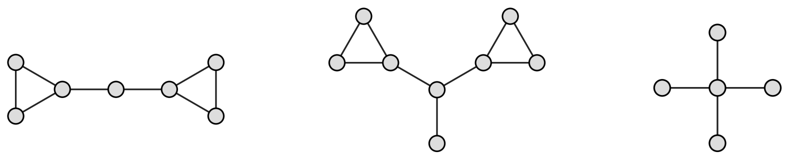


Figure 2: Triangle-Rings of sizes 4, 6 and 8 along with a clustering of the triangle ring of size 8 which matches Lower Bound 3. Some of the edge disjoint P_3s are highlighted in different colors.

Lower Bound 3 *Let G be a graph that contains a triangle-ring G_T of size s as an induced subgraph. To transform G into a cluster graph, at least half of the edges of G_T have to be deleted. Moreover, there are exactly two ways of transforming G_T into a cluster graph while still matching this bound. Both entail deleting the triangles alternatingly.*

Proof. We show the lower bound by showing that there are $1.5s$ edge disjoint P_3s in G_T. This can also be seen in Figure 2. Firstly, consider the cordless cycle that is formed by the inner vertices. As it consists of s edges, there are clearly $0.5s$ edge disjoint P_3s in this cordless cycle. Secondly, consider the other $2s$ edges of G_T, more precisely the edges incident to outer vertices. Each inner vertex forms a P_3 with two outer vertices. Note that these P_3s are edge disjoint from each other and from the P_3s of the cordless cycle. As there are s inner vertices, there are s edge disjoint P_3s within the edges incident to outer vertices. This brings the total to 1.5s edge disjoint P_3s. We therefore know that in G_T at least $1.5s$ edges have to be deleted. This is exactly half of all edges of G_T. As seen in Figure 2, this bound can be matched by deleting the triangles alternatingly. Thus, in any optimal clustering, in each of the described P_3s exactly one edge has to be deleted. There are exactly two ways of transforming the cordless cycle into a cluster graph while deleting exactly one edge in each P_3. For both of these ways we can directly derive an optimal clustering of G_T. Consider an edge $\{v_1, v_2\} \in E$ of the cordless cycle that does not get deleted and thus is part of a cluster. This cluster can only consist of vertices that are adjacent to both the vertices v_1 and v_2. There is only one outer vertex that satisfies this condition. Therefore, all edges incident to v_1 or v_2 that are not part of the triangle $\{v_1, v_2, v_3\}$ have to be deleted. Performing this for all $0.5s$ remaining edges of the cordless cycle deletes s edges and leaves G_T as a cluster graph consisting of s triangles. In total this procedure deletes $0.5s + s = 1.5s$ edges. □



(a) (2,2) (b) (2,3) (c) (0,4)

Figure 3: Triangle-Stars with different signatures.

3.1.2 Triangle-Star

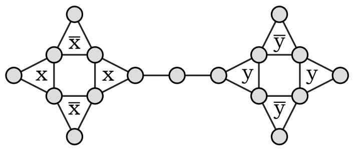
The second structure is called a triangle-star. As seen in Figure 3, it consists of a star and a number of triangles.

Definition 2 (Triangle-Star) *For $t, s \in \mathbb{N}$ with $t \leq s$, the graph $G = (V, E)$ is called a* triangle-star *with signature (t, s) if it can be constructed in the following way:*
1. *Construct an s-star.*
2. *Add t vertex-disjoint triangles so that each of them shares a vertex with one individual non-center vertex of the s-star.*

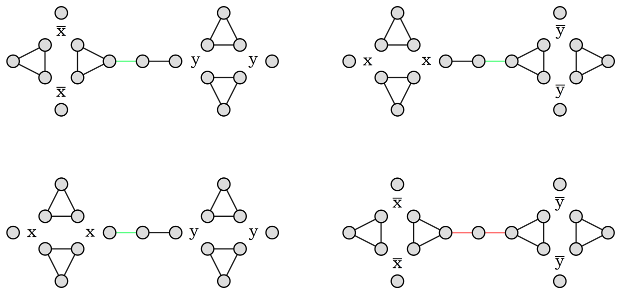
Lemma 1 *Let G be a* triangle-star *with signature (t, s). Let k be the smallest number of edges that have to be deleted to transform G into a cluster graph. We have:*

$$k = \begin{cases} s & \text{if } t = s, \\ s - 1 & \text{if } t < s. \end{cases}$$

Proof. For every triangle we apply Reduction Rule 3. This results in the deletion of t edges. The remaining graph is a star with $s - t$ edges. If $t = s$ we are done and the total cost is equal to s. If $t < s$ we apply Reduction Rule 4, which results in the deletion of $(s - t) - 1$ edges. The total cost is then equal to $s - 1$. □



(a) Constructed Graph.



(b) 4 clusterings for 4 assignments. Observe that one triangle-star requires 2 edge deletions (red), the others only require 1 edge deletion (green)

Figure 4: The formula $(x \vee y)$ as an instance of Cluster Deletion.

Expressing Boolean Formulas as Instances of Cluster Deletion

To show how the special instances defined in the previous sections can be used to model boolean formulas, we give a simple example. We construct a graph which models the clause $(x \vee y)$. It can be seen in Figure 4a. We construct two triangle-rings of size 4 and mark their triangles alternatingly with x and $\overline{x}$ as well as with y and $\overline{y}$ respectively. We then connect a triangle marked with x and a triangle marked with y by a P_3. The reason why this graph models the formula $(x \vee y)$ becomes apparent when considering how an optimal clustering of it looks. By Lower Bound 3, in the triangle-rings at least $2 \cdot 6 = 12$ edges have to be deleted. Moreover, in combination there are exactly $2^2 = 4$ ways of matching this bound by alternatingly deleting the triangles of both of the triangle-rings. Performing any of them would leave a triangle-star with signature $(t, 2)$ where $t \in \{0, 1, 2\}$, as Figure 4b shows. By Lemma 1, this triangle-star has cost 2 if it has signature $(2, 2)$ and cost 1 otherwise. In other words: If performing the edge deletions in the triangle-rings leaves a triangle-star with two triangles, the clustering requires an additional edge deletion and is thus not optimal. Now we take into account the markings of the triangles: The only way for the triangle star to require an additional edge deletion is if the triangles marked with $\overline{x}$ and $\overline{y}$ get deleted. This corresponds to the assignment $\overline{x}$ and $\overline{y}$ which is the only assignment that does not satisfy the clause $(x \vee y)$.

In the following, we sketch how to construct a graph from an arbitrary formula Φ in 3-CNF with the variable set $\mathcal{X}$ and the clause set $\mathcal{C}$. Variables are modeled by triangle-rings. For each $x \in \mathcal{X}$, we construct a triangle-ring and alternatingly mark its triangles with x and $\overline{x}$. The size of the triangle-ring depends on the number of occurrences of the variable x. Clauses are modeled by 3-stars. For each clause $c \in \mathcal{C}$, we add a vertex v_c and add three edges to connect v_c to three triangles that are marked according to the literals of the clause c. This way, if we were to delete the triangles of the triangle-rings alternatingly, the resulting graph does only consist of triangle-stars of signature $(t, 3)$ where $t \in \{0, 1, 2, 3\}$.

We can summarize the central idea behind our reduction as follows: The constructed graph G corresponds to the given formula Φ. The triangle-rings in G correspond to the variables $\mathcal{X}$. For a variable-ring, the two possible clusterings matching Lower Bound 3 correspond to the two boolean values a variable can assume. Deleting the triangles of the triangle-rings alternatingly corresponds to picking an assignment $\alpha : \mathcal{X} \rightarrow \mathbb{B}$. The resulting triangle-stars correspond to the clauses $\mathcal{C}$ after filling in the boolean values according to α but before evaluating the clauses themselves. A triangle getting deleted corresponds to a literal being set to `true`. A triangle not getting deleted and therefore being present in a triangle-star corresponds to a literal being set to `false`. Remember that the signature (t, s) of a triangle-star refers to the number of triangles t and the s-star. Further remember that a triangle-star has a clustering matching Lower Bound 1, if and only if $t < s$. A triangle-star having a signature of $(t, 3)$ where $t \in \{0, 1, 2\}$ corresponds to a clause evaluating to `true`. A triangle-star having a signature of $(3, 3)$ corresponds to a clause evaluating to `false`. Solving Cluster Deletion on a triangle-star corresponds to evaluating a clause. A clustering of G that matches Lower Bound 2+3 corresponds to a satisfying assignment of Φ.

Towards Unit Disk Graphs

When constructing a unit disk graph, we cannot create edges arbitrarily. Implementing the ideas described above to express boolean formulas as instances of Cluster Deletion on unit disk graphs therefore comes with a challenge. One can see that triangle-rings can be created in unit disk graphs by carefully crafting a unit disk representation. However, connecting them directly with 3-stars is only possible for very simple formulas, as the triangle-rings would have to be placed closely together. To overcome this

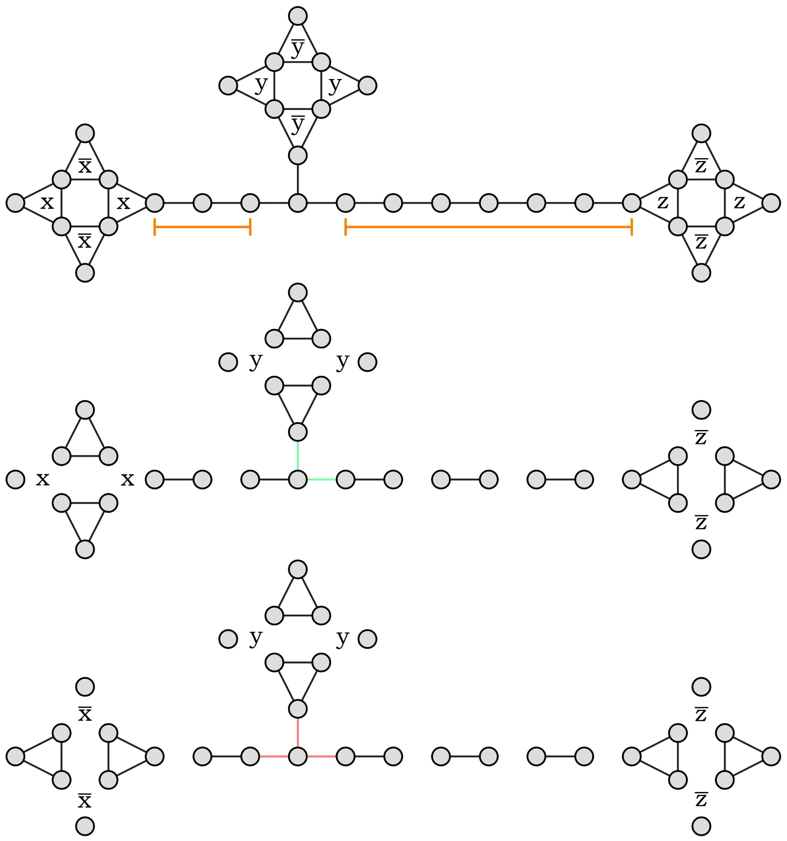


Figure 5: Top: The clause $(x \vee \overline{y} \vee z)$ as an instance of Cluster Deletion on unit disk graphs. Note the cables consisting of 2 and 6 edges (orange). Middle: The satisfying assignment $\{x, y, z\} \rightarrow \{$`true, true, false`$\}$. Bottom: The non-satisfying assignment $\{x, y, z\} \rightarrow \{$`false, true, false`$\}$.

challenge, we make use of Reduction Rule 5. Remember that Reduction Rule 5 allows us to replace an induced P_4 by an edge, if its middle vertices have no other neighbors. Generally speaking, Reduction Rule 5 allows us to shorten the length of paths. Consider the edges of a 3-star as paths consisting of only one edge. In our construction, we extend these paths by adding longer paths of even length, called *cables*. An example of this can be seen in Figure 5. By exhaustive application of Reduction Rule 5, we could shorten the paths until they consist of only one edge again. Note that this application is only theoretical, as the graph would no longer be a unit disk graph. It does, however, ensure that the instance constructed is equivalent.

3.2 Reduction

In this section we present our reduction of 3-SAT to Cluster Deletion on unit disk graphs. We first define a set of gadgets by giving unit disk representations of them. We then explain how to combine these gadgets to construct a unit disk graph G from a given boolean formula Φ in 3-CNF. We also describe how to calculate the critical cost k, which completes the construction of the instance (G, k) of Cluster Deletion. We then prove that (G, k) is equivalent to the instance of 3-SAT implied by Φ. In the last part of this section we bound the size of the graph G which will allow us to bound the running time of Cluster Deletion on unit disk graphs.

3.2.1 Gadgets

In the following, we formally define the gadgets used in our reduction. We do this by showing a unit disk representation of each type of gadget, that is, a set of points in the two-dimensional plane. This way we make sure the gadgets can in fact be created in unit disk graphs. First we do, however, establish a way of reliably connecting different gadgets.

Interface Vertex Connecting two gadgets, as it is often done in reductions, is not trivial when constructing a unit disk graph. For this reason we introduce the notion of an *interface vertex*. Two gadgets that are placed next to each other in the grid depicted in Figure 1 may share an interface vertex. Each interface vertex is defined in two gadgets and serves to connect them. When placing the gadgets in the two-dimensional plane, an interface vertex will be placed twice in exactly the same position. At the end of constructing the unit disk graph, we will identify any two vertices with the same position as the same vertex. In our figures, the interface vertices are marked in red as they are also helpful when understanding how the gadgets are combined.

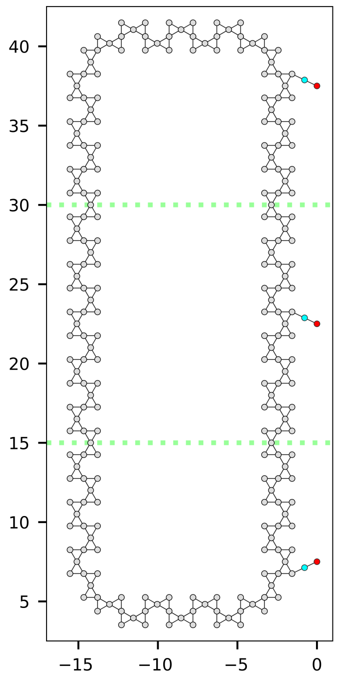


Figure 6: A variable gadget: The border between the different parts is green. The interface vertices are red, the switch vertices are blue. The variable occurs three times, the last occurrence is negated.

Variable Gadget A variable gadget is depicted in Figure 6. It is essentially a large triangle-ring. The size of this triangle-ring depends on the number of times the corresponding variable occurs in the formula Φ. We distinguish between the lower part, middle part and upper part of a variable gadget. Each variable gadget has a lower and an upper part for the first and last time its variable occurs in Φ according to an ordering which will be defined at the beginning of our construction. Furthermore it has one middle part for each additional occurrence. The distance between two adjacent vertices in the variable ring is $\sqrt{\frac{45}{64}} \approx 0.84$. When a variable occurs in a clause it is either negated or not negated. To account for this we use a switch functionality, depicted by the blue vertices in Figure 6. If the occurence in the corresponding clause is negated, then the lower vertex is placed, forming a P_3 from the interface vertex to the lower triangle. If it is not negated, then the upper vertex is placed, forming a P_3 from the interface vertex to the upper triangle.

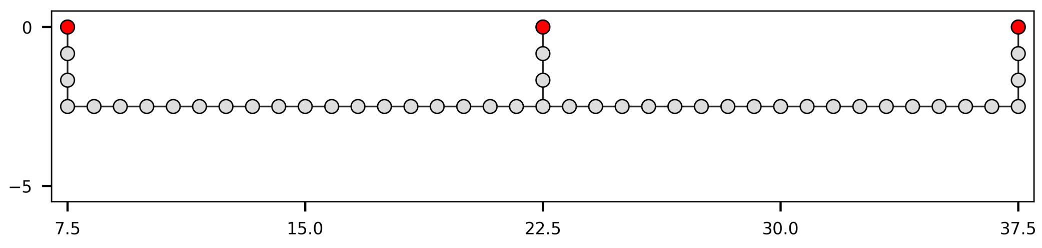


Figure 7: A clause gadget.

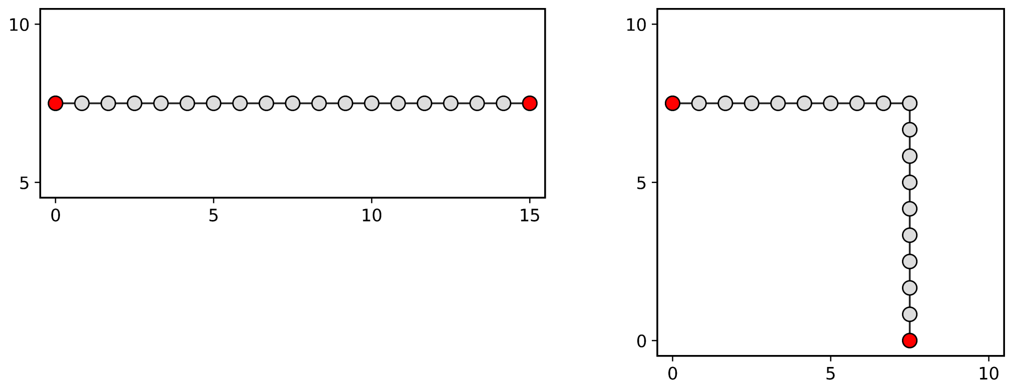
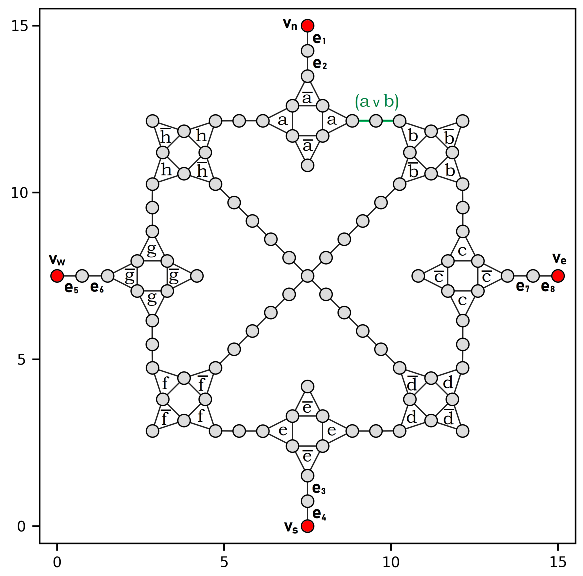


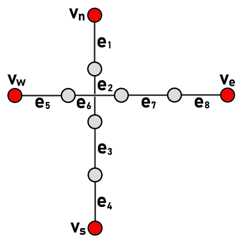
Figure 8: Two cable gadgets.

Clause Gadget A clause gadget is depicted in Figure 7. It consists of three path graphs which meet at a 3-star. The distance between two adjacent vertices is $\frac{15}{18} = 0.8\overline{3}$. This ensures that we do not induce diagonal edges at the right angles, since, according to the Pythagorean theorem, we have $0.8\overline{3}^2 + 0.8\overline{3}^2 > 1^2$. A clause gadget has 3 interface vertices.

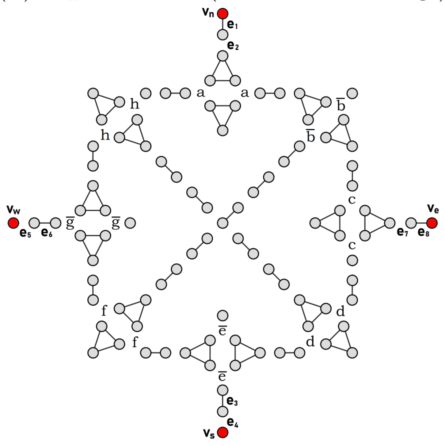
Cable Gadget There are three types of cable gadgets: horizontal, vertical and right-angled. As depicted in Figure 8, all of them are path graphs. The distance between two adjacent vertices is $\frac{15}{18} = 0.8\overline{3}$. This ensures that we do not induce diagonal edges at the right angle, since, according to the Pythogareon theorem, we have $0.8\overline{3}^2 + 0.8\overline{3}^2 > 1^2$. The two vertices at the ends are interface vertices.



(a) unit disk representation, the variable rings (letters) and the clause $(a \vee b)$ (green) are marked for the proof of Lemma 2.



(b) replacement (not a unit disk g.)



(c) clustering

Figure 9: Crossing Gadget

Crossing Gadget Our crossing gadget is inspired by a crossing gadget from a reduction of 3-SAT to Planar 3-SAT [55], where it is used to express 3-CNF formulas in planar 3-CNF. Our crossing gadget is depicted in Figure 9a. It consists of eight triangle-rings and nine connections between them. A crossing gadget has four interface vertices denoted by v_n, v_e, v_s and v_w. Let $E^* = \{e_1, ..., e_8\}$ denote the eight edges closest to the interface vertices. The functionality of a crossing gadget is proven by Lemma 2. It essentially achieves the effect of two cable gadgets at once: a horizontal and a vertical one. Let (G, k) be an instance of Cluster Deletion where G contains a crossing gadget G_C as an induced subgraph and k is calculated by Lower Bound 1-3. We can replace G_C by two P_5s, as depicted in Figure 9b, and reduce k by 67. The edges of the P_5s are denoted by $E^* = \{e_1, ..., e_8\}$ as well. For a clustering to match Lower Bound 1-3, exactly four of the edges in E^* have to be deleted. This holds for the crossing gadget G_C before performing the replacement as well as for the two P_5s.

Lemma 2 *Let G be a graph that contains a crossing gadget G_C as an induced subgraph so that only its four interface vertices v_n, v_e, v_s and v_w are adjacent to $G \backslash G_C$. Let G' be the graph obtained by replacing the crossing gadget G_C in G by a replacement graph G_R that consists of two P_5 from v_n to v_s and from v_w to v_e respectively. There is an optimal solution of* Cluster Deletion *on G that deletes 71 edges in G_C if and only if there is an optimal solution of* Cluster Deletion *on G' that deletes 4 edges in G_R.*

Proof. First we observe that by the Lower Bounds 1-3, to turn G_c into a cluster graph at least 71 of its edges have to be deleted. To avoid a tedious case distinction we make use of the special subgraphs described in Section 3.2 and the way they model boolean formulas. There are eight triangle rings in a crossing gadget, as indicated in Figure 9a. We interpret them as variables that result to `false` if the triangles pointing outwards and inwards get deleted and to `true` otherwise. Observe the variable rings a and b. As suggested in Section 3.1.3, we interpret the path between them as the clause $(a \vee b)$. We do so similarly for all eight variable rings. In total a crossing gadget models nine clauses.

$$(a \vee b) \qquad (b \vee c) \qquad (c \vee d) \qquad (d \vee e) \qquad (e \vee f)$$

$$(f \vee g) \qquad (g \vee h) \qquad (h \vee a) \qquad (\overline{b} \vee \overline{d} \vee \overline{f} \vee \overline{h})$$

Note that the last clause has four literals, as it originates from the 4-star and the paths in the middle of the crossing gadget. By conjoining these clauses and applying a series of boolean reduction rules we see that they imply:

$$(a \vee e) \wedge (c \vee g)$$

To match the lower bound when turning G_C into a cluster graph, both of these clauses need to be satisfied. The clause $(a \vee e)$ tells us that in the variable rings a and e we may only delete one of the triangles pointing at v_n and v_s respectively. It follows that, to match the lower bound, we need to delete either the edge e_2 or the edge e_3. This is the exact functionality of the replacement, where e_2 and e_3 are adjacent in the (v_n, v_s)-path. Similarly, we can show that the clause $(c \vee g)$ implies that the crossing gadget is subject to the same functionality as the (v_w, v_e)-path of the replacement. The edges E', which appear in both the crossing gadget and the replacement, can therefore be considered as linked. For any clustering of G_C that matches the lower bound of 71 edge deletions we can obtain a clustering of G_R with 4 edge deletions where out of E' the same edges get deleted. For any clustering of G_R that matches the lower bound of 4 edge deletions we can obtain a clustering of G_C with 71 edge deletions where out of E' the same edges get deleted. An example of this is depicted in Figure 9c. $\square$

Details Table 1 shows the exact number of vertices and edges in each type of gadget. Additionally, it contains the number of edge deletions necessary to turn the individial gadgets into cluster graphs. These numbers can be calculated using Lower Bound 1-3.

Gadget-Type	Vertices	Edges	k_{min}	Count
Variable (per part)	82	122	61	$3 \cdot \lvert\mathcal{C}\rvert$
Clause	46	45	23	$\lvert\mathcal{C}\rvert$
Cable (right-angled)	19	18	9	$3 \cdot \lvert\mathcal{C}\rvert$
Cable (straight)	19	18	9	$O(\lvert\mathcal{C}\rvert^2)$
Crossing	97	140	71	$O(\lvert\mathcal{C}\rvert^2)$

Table 1: Details of the Gadgets used in the Reduction

3.2.2 Construction

In the following, we formally define how to construct an equivalent instance of Cluster Deletion on unit disk graph when given an instance of 3-SAT.

Graph

Let Φ be a boolean formula in 3-CNF with the variable set $\mathcal{X}$ and clause set $\mathcal{C}$. We first set an arbitrary order for the variables, clauses and literals within clauses. Let L_c be the list of literals sorted by their clause. For an integer i with $1 \leq i \leq |L_c|$ we denote with $L_c[i]$ the literal at position i in clause order. For a literal $\ell \in L_c$ we denote with $L_c^{index}[\ell]$ the position of ℓ in clause order. Let L_v be the list of literals sorted by their variable. For an integer i with $1 \leq i \leq |L_v|$ we denote with $L_v[i]$ the literal at position i in variable order. For a literal $\ell \in L_v$ we denote with $L_v^{index}[\ell]$ the position of ℓ in variable order.

We now create a unit disk graph $G = (V, E)$ by giving a unit disk representation, that is, the coordinates the vertices in V have in the two-dimensional plane. An example of the underlying schematic is given by Figure 1.

For $i, j \in \mathbb{Z}$ with $-1 \leq i, j < 3 \cdot |\mathcal{C}|$ we adress with $tile\langle i, j\rangle$ the area in the two-dimensional plane with $x = [i \cdot 15, (i+1) \cdot 15]$ and $y = [j \cdot 15, (j+1) \cdot 15]$. The gadgets defined in the previous section are placed in the tiles in the following way:

Clauses: For $i \in \mathbb{Z}$ with $0 \leq i < |\mathcal{C}|$ we place a clause gadget spanning the following three tiles: $tile\langle 3i, -1\rangle$, $tile\langle 3i+1, -1\rangle$, $tile\langle 3i+2, -1\rangle$.

Variables: First we place the bottom part of a variable gadget in $tile\langle -1, 0\rangle$ and a top part of a variable gadget in $tile\langle -1, |L_v| - 1\rangle$. Then we place the rest of the variable gadgets: For $i \in \mathbb{N}$ with $1 < i < |L_v|$ let λ_0, λ_1 and λ_2 be the variables of the literals $L_v[i-1]$, $L_v[i]$ and $L_v[i+1]$ respectively. In $tile\langle -1, i-1\rangle$ we place a ...

... top part of a variable gadget	**if** $\lambda_1 \neq \lambda_2$,
... middle part of a variable gadget	**if** $\lambda_1 = \lambda_2$ **and** $\lambda_1 = \lambda_0$,
... bottom part of a variable gadget	**if** $\lambda_1 \neq \lambda_0$.

To take into account whether the literal $L_v[i]$ is negated, we make sure the interface vertex forms a P_3 with the right triangle. If $L_v[i]$ is negated, we place the switch vertex between the interface vertex and the lower triangle. If the literal $L_v[i]$ is not negated, we place the switch vertex between the interface vertex and the upper triangle. Additionally, we alternatingly mark the triangles with λ_1 and $\overline{\lambda_1}$. In total, we essentially place one variable gadget for every variable. All middle parts from a bottom part to the next top part above it belong to one variable gadget.

Connections: For each pair of values $i, j \in \mathbb{Z}$ with $1 \leq i \leq |L_c|$ and $1 \leq j \leq |L_c|$ in $tile\langle i-1, j-1\rangle$ we place a ...

... corner cable gadget	**if** $L_c[i] = L_v[j]$,
... vertical cable gadget	**if** $L_v^{index}[L_c[i]] > j$ **and** $L_c^{index}[L_v[j]] < i$,
... horizontal cable gadget	**if** $L_v^{index}[L_c[i]] < j$ **and** $L_c^{index}[L_v[j]] > i$,
... crossing cable	**if** $L_v^{index}[L_c[i]] > j$ **and** $L_c^{index}[L_v[j]] > i$,
... nothing	**else**.

We essentially model the associated graph of the formula Φ, as depicted in Figure 1. We connect the variable gadgets and the clause gadgets via cable gadgets, as well as crossing gadgets if a tile would contain two cables.

Critical cost k

Together with the graph G, the integer k of allowed edge deletions makes up an instance of Cluster Deletion. As explained in Section 3.1.3, the general idea is that the formula Φ is satisfiable if and only if there exists a clustering that meets Lower Bound 1-3. For the different gadgets, the minimum number of edge deletions necessary to turn them into cluster graphs are listed in Table 2. We keep a count on how many of the different gadgets we place in the construction. The critical cost k is equal to the total number of minimum edge deletions required to turn all gadgets into cluster graphs:

$$k = 9 \cdot \#^{cable}_{gadgets} + 71 \cdot \#^{crossing}_{gadgets} + 61 \cdot (3 \cdot |\mathcal{C}|) + 23 \cdot |\mathcal{C}| \tag{1}$$

$$k = 9 \cdot \#^{cable}_{gadgets} + 71 \cdot \#^{crossing}_{gadgets} + 206 \cdot |\mathcal{C}| \tag{2}$$

3.2.3 Correctness

In the following, we will show the correctness of the reduction described above. The idea of the proof is that we first transform the constructed instance of Cluster Deletion with Reduction Rule 5 and according to Lemma 2. This will result in a graph which only consists of the special subgraphs defined in Section 3.1.

Lemma 3 *For any given formula Φ in* 3-CNF *the reduction described above creates a unit disk graph G and calculates a number k so that the following equivalence holds:*

Φ *is satisfiable.* $\Leftrightarrow$ (G, k) *is a yes-instance of* Cluster Deletion.

Proof. Let (G', k') be the instance of Cluster Deletion equivalent to the instance (G, k) which is is obtained in the following way. We first replace all crossing gadgets in the way suggested in Lemma 2 and lower k by 67 for each crossing gadget replaced. We then apply Reduction Rule 5 exhaustively. This shortens the cable gadgets that connect variable gadgets and clause gadgets. The graph G' now only consists of the triangle rings which are connected by 3-stars. Note that this does not alter the variable gadgets or the 3-stars of the clause gadgets. They are still connected in the same way. Further note that G' is not a unit disk graph. This does not matter since we are only showing the equivalence of instances. We now prove Lemma 3 by showing that the equivalence holds for (G', k'):

($\Rightarrow$) Let α be an assignment of the variable set $\mathcal{X}$ which satisfies Φ. We can turn G' into a cluster graph with exactly k' edge deletions in the following way: First, we delete the triangles of the variable gadgets according to α. More specifically, for each variable gadget, we delete the edges of the triangles that are marked according to the value assigned to its variable in α. This matches Lower Bound 3. Other than components that are already clusters, the remaining graph consists of $|\mathcal{C}|$ triangle-stars, one for each clause gadget placed. Recall that the clause gadgets were connected to variable gadgets via cable gadgets, according to the formula Φ. The length of these connections was reduced to exactly one edge in the beginning of this proof. The triangles of a triangle-star therefore correspond to the literals of the clause its star originated from. If a triangle did not get deleted, its corresponding literal was set to `false`. For the assignment α, in every clause at least one literal is set to true. By deleting the triangles according to α, in every triangle-star, at least one triangle gets deleted. By Lemma 1, all triangle-stars can therefore be transformed into a cluster graph while matching Lower Bound 2. As both lower bounds are matched, we know that G' can be turned into a cluster graph by k' edge deletions.

($\Leftarrow$) We know that (G', k') is a yes-instance of Cluster Deletion. Since k' is calculated according to Lower Bound 1-3 we know that there exists a clustering of G' which matches all three of them. In this clustering, for each triangle ring, the edges of every second triangle are deleted. Performing this deletion on G' would result in a graph consisting of $|\mathcal{C}|$ triangle-stars, one for each clause gadget placed. These triangle-stars can be transformed into a cluster graph while matching Lower Bound 2. This implies that in all resulting triangle-stars at least one triangle got deleted. The triangles of a triangle-star correspond to the literals of the clause its star originated from. As in all resulting triangle-stars at least one triangle got deleted, we know that in all clauses at least one literal is set to true. The existence of this clustering therefore implies the existence of an assignment which satifies all clauses. ■

We have shown that for any given instance of 3-SAT, our reduction creates an equivalent instance of Cluster Deletion. We are now ready to prove Theorem 1. Our reduction is a polynomial time reduction: Creating the unit disk graph and calculating the critical number k clearly takes time polynomial in the size of the formula Φ. If it were possible to solve Cluster Deletion on unit disk graphs in polynomial time, we could solve 3-SAT in polynomial time as well. Since the latter is NP-complete, we obtain the NP-hardness of the former. Finally, we observe that the constructed unit disk graph is planar and has maximum degree 4, justifying the restrictions in Theorem 1. $\square$

3.2.4 Complexity

For a given instance of 3-SAT, the reduction creates an instance of Cluster Deletion. An instance of 3-SAT consists of a formula Φ in 3-CNF with variable set $\mathcal{X}$ and clause set $\mathcal{C}$. An instance of Cluster Deletion consists of a graph G and a number k. In the following, we calculate the asymptotic size of the graph G. More specifically, we bound the number of vertices n and edges m with respect to the number of clauses $|\mathcal{C}|$ in the formula Φ. As shown in Table 2, all gadgets have a constant number of vertices and edges. The size of G therefore only depends on the number of gadgets placed in the construction. Clearly, there are $|\mathcal{C}|$ clause gadgets. We count the variable gadgets by their individual parts: in total there are $3 \cdot |\mathcal{C}|$ individual parts, as for each occurence of a variable in Φ, exactly one part is placed. The number of cable gadgets and crossing gadgets depends on the ordering we pick in the construction. We can, however, bound their number by taking into account the area they are placed in. As shown in Figure 1, the area covered by cable and crossing gadgets is the square defined by the variable gadgets and the clause gadgets. Both the number of cable gadgets and the number of crossing gadgets thus scale quadratically with the number of clauses. We can therefore conclude that the constructed graph G has $n = O(|\mathcal{C}|^2)$ vertices and $m = O(|\mathcal{C}|^2)$ edges.

ETH We are now ready to prove Theorem 2. The lower bound follows from the size of the graph created in the reduction. For a given formula Φ the graph constructed has $n = O(|\mathcal{C}|^2)$ vertices and $m = O(|\mathcal{C}|^2)$ edges. If there was an algorithm solving Cluster Deletion on unit disk graphs in $2^{o(\sqrt{n+m})}$ time, there would also be an algorithm solving 3-SAT in $2^{o(|\mathcal{C}|)}$ time. This is not possible unless the ETH fails. Again, the structural properties of the constructed graph G justify the restrictions in Theorem 2. $\square$

3.3 Implications for Cluster Editing

By reduction from 3-Sat, we have shown the NP-hardness of Cluster Deletion on unit disk graphs. Assuming the exponential time hypothesis (ETH), we have also shown a running time lower bound. A problem closely related to Cluster Deletion is Cluster Editing. Besides edge deletions, Cluster Editing also allows the addition of edges to transform the input graph into a cluster graph.

Corollary 1 Cluster Editing *is NP-hard even on planar unit disk graphs with maximum degree* 4*. Furthermore,* Cluster Editing *cannot be solved in* $2^{o(\sqrt{n+m})}$ *time on planar unit disk graphs with maximum degree* 4*, unless the ETH fails.*

Proof. On diamond-free graphs, there is always an optimal solution of Cluster Editing that does not require the addition of edges [61]. In other words, on diamond-free graphs, the problem of Cluster Editing is equivalent to Cluster Deletion. The unit disk graph G constructed in our reduction is diamond-free. An algorithm solving Cluster Editing on G would also solve Cluster Deletion on G. We conclude that both the NP-hardness result of Theorem 1 and the running time lower bound of Theorem 2 also apply to Cluster Editing. □

Conclusion

In this chapter, we have shown the NP-hardness of both Cluster Deletion and Cluster Editing on planar unit disk graphs of maximum degree 4. To the best of our knowledge this is the first proof of NP-hardness for Cluster Editing on graphs of maximum degree 4. To accompany this result, we have shown that both Cluster Deletion and Cluster Editing cannot be solved in $2^{o(\sqrt{n+m})}$ time on planar unit disk graphs with maximum degree 4, unless the exponential time hypothesis (ETH) fails. The form of the running time lower bound, namely the square root in the exponent, is consistent with many other running time lower bounds for problems on unit disk graphs [32, 9, 28]. To ensure that our running time lower bound is tight, one would have to show that an algorithm with a matching running time exists.

4 Parameterized Complexity

In this chapter we will study the complexity of Cluster Deletion parameterized by treewidth ω. In Section 4.1 we present an FTP-algorithm and analyze its running time and space requirements. In Section 4.2 we conclude what our results imply for the complexity of Cluster Deletion on graph classes of bounded treewidth, such as planar graphs. In Section 4.3 we discuss how our algorithm can be adapted to solve related problems such as Clique Partition instead. Additionally, we elaborate on why adressing Cluster Editing with similar adaptations might not be straightforward.

4.1 Cluster Deletion parameterized by Treewidth

Courcelle's Theorem states that any problem definable in MSO_2 can be solved in linear time on graphs of bounded treewidth [24]. This implies the existence of an algorithm with a running time of $O(f(\omega) \cdot n)$ whereas the exact form of f depends on the problem at hand. Since Cluster Deletion can be expressed in MSO_2 [50], it is FPT for the parameter treewidth. It is, however, an open question whether such an algorithm admits to a single exponential running time [46], as it has yet to be formulated in concrete terms.

Theorem 3 Cluster Deletion *can be solved in* $2^{O(\omega)} \cdot \omega \cdot n$ *time on graphs of treewidth* ω.

For a given graph of treewidth ω it is possible in $2^{O(\omega)} \cdot n$ time to create a tree decomposition of width at most $2\omega + 1$ [52]. This tree decomposition can be turned into a nice tree decomposition in $O(\omega^2 \cdot n)$ time [2]. Observe that both steps take linear time on graphs with bounded treewidth and are thus suitable to be part of an algorithm in the sense of Courcelle's Theorem. If a nice tree decomposition would help in solving Cluster Deletion in a similar running time, the combined algorithm would prove Theorem 1.

Lemma 4 Cluster Deletion *can be solved in* $O(3^{\omega} \cdot \omega \cdot n)$ *time provided a nice tree decomposition of width* ω *is given.*

In the remainder of this section we prove Lemma 2, and thus Theorem 1, by presenting a dynamic algorithm that solves Cluster Deletion by traversing a nice tree decomposition.

A vertex set is called a separator if the graph would have multiple connected components without it. A common algorithmic strategy is to consider all possibilities of dividing the sepeartor set among the components and then solving the problem independently for each of them. Like most graph problems, Cluster Deletion can be solved independently on connected components. Two vertices from different connected components cannot be in the same cluster since they are not adjacent. Each vertex of the separator can form a cluster with vertices from only one of the connected components.

One way of reliably finding separators is with the help of a nice tree decomposition. For each node, the vertices of its bag constituate a separator which divdes the other vertices into those which appear above and those which appear below the node. Our algorithm follows the usual procedure of dynamic algorithms on nice tree decompositions. We start at the leaf nodes and combine our way up, storing the already calculated solutions. At a node our algorithm considers all ways of splitting the vertices of its bag into those that form clusters with vertices from above and those which form clusters with vertices from below. For each split we then calculate the solution of Cluster Deletion on the lower component.

4.1.1 Dynamic Algorithm

Let $G = (V, E)$ be a graph. Let D be a nice tree decomposition of G with width ω. For a node i of D we denote with $V_i \subseteq V$ the vertices contained in its bag. With $G_i \subseteq G$ we denote the graph that is induced by the vertices V_i and those that appear below i in D.

We traverse D in post-order: For each node i we create a table T_i which stores an integer for each possible subset $S \subseteq V_i$. Formally $T_i : \mathcal{P}(V_i) \to \mathbb{Z}$. The values of T_i are calculated as described in the rules below, depending on the type of the node i. They satisfy the following equation:

$$\begin{aligned} T_i[S] \quad = \quad & \text{number of edges remaining after solving} \\ & \text{Cluster Deletion on the graph } G_i - (V_i \backslash S). \end{aligned}$$

By definition the root of a nice tree decomposition is empty. In other words, we have $V_{\text{root}} = \emptyset$. Therefore, the table T_{root} has only one entry, for the empty set. Since all vertices are in bags below, we have $G_{\text{root}} = G$. By applying this to the equation above, we conclude that after traversing D we have:

$$\begin{aligned} T_{\text{root}}[\emptyset] \quad = \quad & \text{number of edges remaining after solving} \\ & \text{Cluster Deletion on the graph } G. \end{aligned}$$

Recurrence Formula.

In the following, we define the recurreces that assure that the equation stated above is satisfied for each table entry. The formula is different depending on the type of node.

Leaf Node.

$$T_{\text{leaf}}[\emptyset] = 0$$

In nice tree decompositions, the bags of leaf nodes are empty. In other words, we have $V_{\text{leaf}} = \emptyset$. Therefore, the table T_{leaf} has only one entry, for the empty set. Since all vertices are in bags above, the graph G_{leaf} has no vertices. On the empty graph, the optimal solution of Cluster Deletion does not have any edges. To implement this, for every leaf node we create a table with one entry and set it to zero:

Introduce Node.

$$T_i[S] = \begin{cases} T_j[S] & \text{if } v \notin S, \\ \\ \max\limits_{C \subseteq S\colon\, v \in C,\, C \text{ is clique}} T_j[S \backslash C] + \binom{|C|}{2} & \text{if } v \in S. \end{cases}$$

Claim: Let i be an introduce node which introduces the vertex v above node j. The recurrence formula above assures that for any possible subset $S \subseteq V_i$ the value $T_i[S]$ is equal to the number of edges remaining after solving Cluster Deletion on the graph $G_i - (V_i \backslash S)$.

Proof. We have $V_i = V_j \cup \{v\}$ and $G_i = G_j \cup \{v\}$. We know that the value $T_j[S]$ is equal to the number of edges remaining after solving Cluster Deletion on the graph $G_j - (V_j \backslash S)$. If $v \notin S$, the graph $G_i - (V_i \backslash S)$ is equal to the graph $G_j - (V_j \backslash S)$. Therefore, the values $T_i[S]$ and $T_j[S]$ are equal as well. If $v \in S$, the graph $G_i - (V_i \backslash S)$ includes v. Let $C \subseteq S$ be the vertices which make up the cluster of v. We can think of the graph $G_i - (V_i \backslash S)$ as consisting of C and the rest-graph $G_{\text{rest}} = G_i - (V_i \backslash (S \backslash C))$. By the equalities above, we have $G_{\text{rest}} = G_j - (V_j \backslash (S \backslash C))$. We know that the number of edges remaining after solving Cluster Deletion on G_{rest} is equal to the value $T_j[S \backslash C]$. Therefore, the value $T_i[S]$ is equal to the number of edges in C plus the value $T_j[S \backslash C]$. Since we do not know which vertices, other than v, make up C, we consider all possibilities where C is a clique in G. The value $T_i[S]$ is equal to the highest number of edges which arises from any combination of a clique C and the solution of Cluster Deletion on G_{rest}. ■

Forget Node.

$$T_i[S] = T_j[S \cup \{v\}]$$

Claim: Let i be a forget node which forgets the vertex v from node j. The recurrence formula above assures that for any possible subset $S \subseteq V_i$ the value $T_i[S]$ is equal to the number of edges remaining after solving Cluster Deletion on the graph $G_i - (V_i \backslash S)$.

Proof. We have $V_j = V_i \cup \{v\}$ and $G_j = G_i$. We know that the value $T_j[S \cup \{v\}]$ is equal to the number of edges remaining after solving Cluster Deletion on the graph $G_j - (V_j \backslash (S \cup \{v\}))$. By using the equalities above we see that this graph is equal to the graph $G_i - (V_i \cup \{v\} \backslash (S \cup \{v\}))$, which is equal to the graph $G_i - (V_i \backslash S)$. ■

Join Node.

$$T_i[S] = \max_{\substack{S_j, S_\ell \subseteq S: \\ S_j \cap S_\ell = \emptyset \\ S_j \cup S_\ell = S}} T_j[S_j] + T_\ell[S_\ell]$$

Claim: Let i be a join node which combines the nodes j and ℓ. The recurrence formula above assures that for any possible subset $S \subseteq V_i$ the value $T_i[S]$ is equal to the number of edges remaining after solving Cluster Deletion on the graph $G_i - (V_i \backslash S)$.

Proof. We have $V_i = V_j = V_\ell$ and $G_i = G_j \cup G_\ell$. Let $S_j, S_\ell \subseteq S$ be two sets with $S_j \cap S_\ell = \emptyset$ and $S_j \cup S_\ell = S$. In other words, we split S into two sets S_j and S_ℓ. We can think of a clustering of the graph $G_i - (V_i \backslash S)$ as consisting of two parts: a clustering of the graph $G_j - (V_j \backslash S_j)$ and a clustering of the graph $G_\ell - (V_\ell \backslash S_\ell)$. The number of edges remaining after solving Cluster Deletion on the graphs $G_j - (V_j \backslash S_j)$ and $G_\ell - (V_\ell \backslash S_\ell)$ are equal to the values $T_j[S_j]$ and $T_\ell[S_\ell]$, respectively. Other than that they are complementary with respect to S, we do not know the compositions of the sets S_j and S_l. We therefore consider all possible splits. The number of edges remaining after solving Cluster Deletion on the graph $G_i - (V_i \backslash S)$

is the maximum value of $T_j[S_j] + T_\ell[S_\ell]$ over all possible splits of S into the sets S_j and S_ℓ. ■

4.1.2 Termination

In the following, we briefly ensure that our algorithm terminates. The table of each leaf node is set to have exactly one entry: the empty set is mapped to zero. For every other node, the values of its table depend only on the tables of its child nodes. When traversing a tree in post-order, a partent node is not visited until all its child nodes have been visited. Our algorithm will therefore fill the table of each node, ultimately arriving at the root.

4.1.3 Running Time

In the following, we analyze the running time of our algorithm. We first calculate the running time of creating the table of a single node, depending on its type. The total running time of our algorithm then depends on the number of nodes in the tree decomposition.

Leaf Node. The table of a leaf node can be created in constant time.

Introduce Node. Let the introduce node i introduce the vertex v above node j. To optimize the running time we first create a lookup table with information about the existence of cliques on the graph induced by V_i. For every possible subset $C \subseteq V_i$ we store a boolean value which indicates whether C is a clique. This lookup table allows us to check the existence of a clique in constant time. Creating it takes $2^{|V_i|} \cdot |V_i|^2$ time.

We then fill the table T_i according to the recurrence formula. There are $2^{|V_i|}$ entries in the table T_i, one for each possible subset $S \subseteq V_i$. We make a case-distinction for whether v is part of the set S:
Case $v \notin S$: To calculate the value $T_i[S]$ we query the table T_j for a single value. This is possible in constant time. Since they make up exactly half of the table, calculating all entries with $v \notin S$ takes $2^{|V_i|-1}$ time in total.
Case $v \in S$: To calculate the value $T_i[S]$ we consider all possible clusters which could be formed with v. To avoid lengthy forumlas, we use $S' = S \backslash \{v\}$ and $V_i' = V_i \backslash \{v\}$ as the vertex v is part of all clusters considered. There are $2^{|S'|}$ potential clusters to consider; one for each possible subset $C' \subseteq S'$. We

can express the total number of potential clusters considered to fill all entries with $v \in S$ by the following sum, where $|S'| = x$:

$$\sum_{S' \subseteq V_i'} 2^{|S'|} = \sum_{x=0}^{|V_i'|} \binom{|V_i'|}{x} \cdot 2^x = 3^{|V_i'|} = 3^{|V_i|-1}$$

We can rewrite the first sum by counting the potential clusters we need to consider depending on the size of the set S'. For the case $v \in S$ there are $\binom{|V_i'|}{x}$ table entries with $|S'| = x$. The equivalence holds since for sets of the same size we need to check the same number of potential clusters. After application of the binomial theorem we see that we consider $3^{|V_i|-1}$ potential clusters in total. This is also intuitive, regarding the fact that for every vertex $u \in V_i'$ there are exactly 3 options:

1.	$u \in C$	u is part of the cluster of v.
2.	$u \in S \land u \notin C$	u is part of the partial solution corresponding to $T_i[S]$, but not part of the cluster of v.
3.	$u \notin S$	u is not part of the partial solution corresponding to $T_i[S]$.

For each potential cluster C' we check whether the clique exists in G. This is possible in constant time since we have previously created a lookup table. If C' is in fact a clique we query the table T_j for a single value to calculate the number of edges in the combination. This is possible in constant time aswell. Thus, calculating all entries with $v \in S$ takes $3^{|V_i|-1}$ time in total.

In conclusion: Creating the lookup table and filling the table T_i takes $2^{|V_i|} \cdot |V_i|^2 + 2^{|V_i|-1} + 3^{|V_i|-1}$ time. Recall that the width ω of a nice tree decomposition is defined as the maximum number of vertices in a bag minus one: $\omega = \max_{\iota \in D} |V_\iota| - 1$. It therefore takes $O(3^\omega)$ time to create and fill the table of an introduce node.

Forget Node. Let i be a forget node with the child node j. The table T_i has $2^{|V_i|}$ entries. For each of them, we query the table T_j for a single value. This is possible in constant time. Therefore, filling the table of a forget node takes $2^{|V_i|}$ time in total. With the definition $\omega = \max_{\iota \in D} |V_\iota| - 1$ we can bound this to $O(2^\omega)$.

Join Node. Let the join node i combine the nodes j and ℓ. There are $2^{|V_i|}$ entries in the table T_i, one for each possible subset $S \subseteq V_i$. To calculate the value $T_i[S]$ we consider $2^{|S|}$ possible ways of splitting S into S_j and S_ℓ. We can express the total number of splits considered to fill all entries by the following sum, where $|S| = x$.

$$\sum_{S \subseteq V_i} 2^{|S|} = \sum_{x=0}^{|V_i|} \binom{|V_i|}{x} \cdot 2^x = 3^{|V_i|}$$

We can rewrite the first sum by counting the splits considered depending on the size of the set S. There are $\binom{|V_i|}{x}$ table entries with $|S| = x$. The equivalence holds since for sets of the same size we have to consider the same number of splits. After application of the binomial theorem we see that, to fill all entries, we have to consider $3^{|V_i|}$ splits in total. This is again intuitive, regarding the fact that for every vertex $u \in V_i$ there are exactly 3 options:

1. $u \in S_j$ u is used in the node j.

2. $u \in S_\ell$ u is used in the node ℓ.

3. $u \notin S$ u is not part of the partial solution corresponding to $T_i[S]$.

For each split we query the tables T_j and T_ℓ for one value each and add them. This is possible in constant time. Therefore, filling the table of a join node takes $3^{|V_i|}$ time. With the definition $\omega = \max_{\iota \in D} |V_\iota| - 1$ we can bound this to $O(3^\omega)$.

Total running time A nice tree decomposition, when obtained in the way described in Section 2.3, has at most $O(\omega \cdot n)$ nodes in total. With the exception of forget nodes, there are no evident constrains of how often the different types of nodes can occur.

When traversing D in post-order we visit each node exactly once. Upon visiting a node i we create and fill the table T_i according to the recurrence formula. The running times of that are analyzed above and depicted in Table 3. We conclude that our algorithm takes $O(3^\omega \cdot \omega \cdot n)$ time in total. □

Type of Node	Count	Time
Leaf Node	$O(\omega \cdot n)$	$O(1)$
Introduce Node	$O(\omega \cdot n)$	$O(3^\omega)$
Forget Node	n	$O(2^\omega)$
Join Node	$O(\omega \cdot n)$	$O(3^\omega)$

Table 2: Count & Running Time for different types of nodes.

4.1.4 Space Complexity

In the following, we analyze the space requirements of our algorithm. This essentially consists of bounding the size of the tables created. The table of any node i has $2^{|V_i|}$ entries. An entry consists of a subset $S \subseteq V_i$, which takes $|V_i|$ bits to specify, and an integer for the number of edges in the corresponding partial solution, which takes at most $\log_2 |E|$ bits to store. The table of a node therefore takes up $2^{|V_i|} \cdot (|V_i| + \log_2 |E|)$ bits of space. Again, we assume that any given nice tree decomposition has at most $O(\omega \cdot n)$ nodes. By multiplication we discover that our algorithm has a space complexity of $O(2^\omega \cdot (\omega + \log_2 m) \cdot \omega \cdot n)$.

Note that in a practical application we do not store the tables all at once. The table of a node is only queried when creating the table of its parent node. After that, we can discard it. While useful in practice, it is not evident whether this leads to an asymptotically better space complexity. Of course this only applies as long as we are interested in the solution of Cluster Deletion as the number of edges and not in a specific composition of the clusters.

Conclusion

To prove Lemma 2, we have presented an algorithm that solves Cluster Deletion in $O(3^\omega \cdot \omega \cdot n)$ time when provided with a nice tree decomposition of width ω. As discussed in the beginning of this section, creating such a nice tree decomposition is possible in $2^{O(\omega)} \cdot n$ time. The existence of our algorithm therefore also proves Theorem 1. $\square$

4.2 Direct Implications

With the algorithm presented in the previous section, it is now possible to make statements about the running time of Cluster Deletion on graph classes with bounded treewidth.

Corollary 2 Cluster Deletion *can be solved in* $2^{O(\sqrt{n})} \cdot n^{1.5}$ *time on planar graphs.*

Proof. The treewidth ω of planar graphs is known to be $O(\sqrt{n})$. The most recent bound states that for any planar graph we have $\omega < 3.182\sqrt{n}$ [34]. If we combine this result with Theorem 3 we see that our algorithm solves Cluster Deletion on planar graphs in $2^{O(\sqrt{n})} \cdot n^{1.5}$ time. □

Corollary 3 Cluster Deletion *can be solved in* $2^{O(\sqrt{n})} \cdot n^{1.5}$ *time on unit disk graphs with maximum degree bounded by a constant.*

Proof. Let G be a unit disk graph with maximum degree Δ. The treewidth ω of G can be bounded as $\omega \in O(\Delta^3\sqrt{n})$ [33]. If we assume that the maximum degree Δ is constant, by Theorem 3, we can solve Cluster Deletion on G in $2^{O(\sqrt{n})} \cdot n^{1.5}$ time. □

We observe that the running times stated by Corollary 2 and Corollary 3 both match the running time lower bound implied by Theorem 2 up to a polynomial factor. The algorithm presented in Section 4.1 is therefore the best we can hope for when it comes to solving Cluster Deletion efficiently on both planar graphs and unit disk graphs of constant maximum degree.

4.3 Adaptions for related problems

In this section we discuss how to adapt our algorithm for Cluster Deletion to solve related problems instead. To solve the problems of Minimum Clique Partition or Maximum μ-Clique Packing only minor changes to the recurrence formula are required. Note that we provide these adaptations for demonstrational purposes only, as the running time of $O(3^\omega \cdot \omega \cdot n)$ is not optimal for these problems. In fact, there are dynamic algorithms for both problems that achieve a running time of $O(2^\omega \cdot poly(\omega) \cdot n)$ provided a tree decomposition of width ω is given [68].

4.3.1 Clique Partition

The Minimum Clique Partition problem seeks to partition the vertex set of a graph into a minimum number of disjoint cliques.

Corollary 4 Minimum Clique Partition *can be solved in* $O(3^\omega \cdot \omega \cdot n)$ *time provided a nice tree decomposition of width* ω *is given.*

Proof. We show that the algorithm presented in Section 4.1.1 to solve Cluster Deletion can be adapted to solve Minimum Clique Partition instead. For any node i the values of the table T_i are calculated so that for each subset $S \subseteq V_i$ they satisfy the following equality:

$$T_i[S] \quad = \quad \text{minimum number of vertex-disjoint cliques required to partition the graph } G_i - (V_i \backslash S).$$

To implement this, we redefine the recurrence formula for the introduce node and the join node in the following way. For the leaf node and the forget node the recurrence formula is defined in the same way it was in Section 4.1.1.

(*Introduce Node*:)

$$T_i[S] = \begin{cases} T_j[S] & \text{if } v \notin S, \\ \min\limits_{C \subseteq S:\ v \in C,\ C \text{ is clique}} T_j[S \backslash C] + 1 & \text{if } v \in S. \end{cases}$$

(*Join Node*:)

$$T_i[S] = \min_{\substack{S_j, S_\ell \subseteq S: \\ S_j \cap S_\ell = \emptyset \\ S_j \cup S_\ell = S}} T_j[S_j] + T_\ell[S_\ell]$$

Instead of maximizing the number of edges within the cliques we minimize the number of cliques it takes to partition the vertex set of the graph. For an introduce node we still consider all possible cliques $C \subseteq S$ which include v. Instead of calculating the combination of C and the rest graph G_{rest} that has the most edges, we calculate the combination with the fewest cliques. For a join node we consider all possible splits and calculate the minimum number of cliques required to partition both corresponding graphs. For both types of nodes the recurrence formula can be proven with the same arguments as given in Section 4.1.1. Additionally, the running time of the adapted algorithm is clearly equivalent to that stated in Section 4.1.3. □

4.3.2 Clique Packing

For an integer μ the Maximum μ-Clique Packing problem asks for a maximum number of disjoint μ-cliques, that is, disjoint cliques of size μ.

Corollary 5 Maximum μ-Clique Packing *for* $\mu \geq 3$ *can be solved in* $O(3^\omega \cdot \omega \cdot n)$ *time provided a nice tree decomposition of width* ω *is given.*

Proof. We show that the algorithm presented in Section 4.1.1 to solve Cluster Deletion can be adapted to solve Maximum μ-Clique Packing instead. For any node i the values of the table T_i are calculated so that for each subset $S \subseteq V_i$ they satisfy the following equality:

$$T_i[S] \quad = \quad \begin{array}{l}\text{maximum number of vertex-disjoint} \\ \mu\text{-cliques in the graph } G_i - (V_i \backslash S).\end{array}$$

To implement this, we redefine the recurrence formula of the introduce node in the following way. For the other three types of nodes the recurrence formula is defined in the same way it was in Section 4.1.1.

(*Introduce Node*:)

$$T_i[S] = \begin{cases} T_j[S] & \text{if } v \notin S, \\ \max\limits_{C \subseteq S:\ v \in C,\ C \text{ is clique, } |C| \in \{1, \mu\}} T_j[S \backslash C] + \left\lfloor \dfrac{|C|}{\mu} \right\rfloor & \text{if } v \in S. \end{cases}$$

Instead of maximizing the number of edges within the cliques we maximize the number of cliques of size μ. For an introduce node we consider all possible cliques $C \subseteq S$ that include v and have size μ. We additionally allow the clique $C = \{v\}$ to take into account the case where the vertex v does not end up in a μ clique. The recurrence formula can be proven with the same arguments as given in Section 4.1.1. The running time of the adapted algorithm is clearly not bigger than that stated in Section 4.1.3. $\square$

As Maximum μ-Clique Packing is a generalization of several other clique packing problems we also obtain running time statements for the special cases of it.

Corollary 6 Partition into μ-Cliques *for* $\mu \geq 3$ *and* Partition into Triangles *can be solved in* $O(3^\omega \cdot \omega \cdot n)$ *time provided a nice tree decomposition of width* ω *is given.*

Proof: The running times follow directly from Corollary 5. For a graph $G = (V, E)$ there exists a Partition into μ-Cliques if and only if the solution of Maximum μ-Clique Packing is is equal to $\frac{|V|}{\mu}$. Furthermore, Partition into Triangles is the special case where $\mu = 3$. □

Cluster Editing

The problem of Cluster Editing asks for a minimum number of edge modifications, that is, additions and deletions, to turn a graph into a cluster graph. Naturally, we could approach it in the same way that we approached the other problems. However, when trying to modify the algorithm to solve Cluster Editing instead, we run into a problem. Since Cluster Editing also allows adding edges to turn the graph into a cluster graph, not every potential cluster is already a clique. As a simple example, consider a diamond graph, which has four vertices that induce five edges. It can be turned into a cluster graph by adding a single edge. An optimal tree decomposition of a diamond graph would, however, only contain two bags of size three. We see that not every potential cluster can be found in a single bag. Simple changes to the recurrence formula, which are sufficient to solve other problems, do not seem enough. The running time of Cluster Editing on graphs of bounded treewidth therefore remains an open question.

Conclusion

We see that only small changes to the recurrence formula of our algorithm for Cluster Deletion are sufficient to solve several other graph clustering problems instead. This highlights the versatility of dynamic programming approaches, as well as the similarity of graph clustering problems that appear to be quite different at first glance.

5 Conclusion

In this chapter, we provide a summary of our results and point out open questions that offer potential for future research endeavors.

5.1 Summary

In Chapter 3, we studied the complexity of Cluster Deletion and Cluster Editing on unit disk graphs. We presented a polynomial-time reduction of 3-Sat to Cluster Deletion on unit disk graphs. We observed that the constructed graph is planar, has a maximum degree of 4 and, most importantly, that it is diamond-free. The last observation led us to the conclusion that our reduction is also a reduction to Cluster Editing. Therefore, we have proven that both Cluster Deletion and Cluster Editing remain NP-hard even on planar unit disk graphs with maximum degree 4. To the best of our knowledge this is the first proof of NP-hardness for Cluster Editing on graphs of maximum degree 4. Accompanying this result, we have shown that both Cluster Deletion and Cluster Editing cannot be solved in $2^{o(\sqrt{n+m})}$ time on planar unit disk graphs with maximum degree 4, unless the exponential time hypothesis (ETH) fails.

In Chapter 4, we studied the parameterized complexity of Cluster Deletion for the parameter treewidth ω. We presented a dynamic algorithm that solves Cluster Deletion by traversing a nice tree decomposition. A running time analysis showed that, provided a nice tree decomposition of width ω is given, Cluster Deletion can be solved in $O(3^\omega \cdot \omega \cdot n)$ time. This answers the question of whether such an algorithm admits a single exponential running time, recently mentioned by Italiano et al. [46]. We then considered the direct implications of our result on the complexity of Cluster Deletion on graph classes of bounded treewidth. We concluded that Cluster Deletion can be solved in $2^{O(\sqrt{n})} \cdot n^{1.5}$ time on planar graphs as well as on unit disk graphs with constant maximum degree. This running time matches the lower bound we had stated in Chapter 3 up to a polynomial factor. For demonstrational purposes, we showed how our algorithm can be adapted to solve related graph clustering problems such as Minimum Clique Partition or Maximum μ-Clique Packing.

5.2 Future Work

In this work, we have provided a subexponential algorithm and a matching running time lower bound. However, when taking a closer look at the results, we see that there is still room for improvement.

Firstly, as stated by Theorem 2, our running time lower bound applies to unit disk graphs which are both planar and have maximum degree 4. On the unrestricted class of unit disk graphs, a stronger running time lower bound for Cluster Editing and Cluster Deletion might be possible. The restrictions in Theorem 2 stem from the properties of the graph created in our reduction. To result in a running time lower bound stronger than $2^{o(\sqrt{n+m})}$, a reduction to Cluster Deletion, would have to construct a non-planar unit disk graph without constant maximum degree. This is evident when considering the results of Chapter 4, more specifically Corollary 2 and Corollary 3. Secondly, a subexponential algorithm for Cluster Deletion on unit disk graphs might be possible. As discussed in Section 1.1, there are many problems with subexponential algorithms on unit disk graphs. Of particular interest for this undertaking might be the two recently presented frameworks mentioned [28, 58].

Another possible line of work is a parameterization of clustering problems on unit disk graphs by domain area. The domain area of a unit disk graph refers to the size of the square region its points occupy in the two-dimensional plane. Both Hamilton Cycle and 3-Coloring are FPT for the domain area of a unit disk graph [47]. Although this parameter is native to unit disk graphs and geometric intersection graphs in general, it seems to have been largely unexplored. For Cluster Editing and Cluster Deletion the domain area of a given unit disk graph might be related to the number of clusters in the resulting cluster graph. This might be helpful since, if we require the resulting cluster graph to have exactly p clusters, Cluster Editing can be solved in $O(2^{O(\sqrt{pk})} + n + m)$ time [31].

One of the most famous clustering algorithms is called k-means. In the field of unsupervised machine learning, it is mostly known for its heuristical application, as finding an optimal k-means clustering is NP-hard even in the two-dimensional plane [60]. One idea to cope with the complexity of Cluster Deletion on unit disk graphs is by using an approximation. In the general case, Cluster Deletion is NP-hard to approximate to within some constant factor [64]. A straightforward 2-approximation of Cluster Deletion can be achieved by recursively finding, isolating and removing the biggest clique until the graph is empty [29]. In the general case, this procedure is not possible in polynomial time, as finding the biggest clique is known to be NP-hard. However, finding the biggest clique in a unit disk graph is possible in $O(n^{3.5} \cdot \log(n))$ time [16, 63]. It is therefore possible in $O(n^{4.5} \cdot \log(n))$ time to compute a 2-approximation for Cluster Deletion on unit disk graphs. The natural question is that of whether a better or faster approximation is possible. A recently published framework for the design of efficient polynomial time approximation schemes (EPTAS) on disk graphs might be of particular interest in this undertaking [59].

Lastly, it appears that no FPT-algorithm for Cluster Editing parameterized by treewidth has been presented yet. In Section 4.3 we briefly pointed out the challenge such an algorithm, if it exists, would need to overcome. If such an algorithm were to admit a single exponential running time, it would result in ETH-tight algorithms for Cluster Editing on planar graphs as well as unit disk graphs of constant maximum degree.

References

[1] Jochen Alber and Jiri Fiala. Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *Journal of Algorithms*, 52(2):134–151, 2004.

[2] Ernst Althaus and Sarah Ziegler. Optimal tree decompositions revisited: A simpler linear-time fpt algorithm. *Graphs and Combinatorial Optimization: from Theory to Applications: CTW2020 Proceedings*, pages 67–78, 2021.

[3] 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.

[4] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[5] Aistis Atminas and Viktor Zamaraev. On forbidden induced subgraphs for unit disk graphs. *arXiv preprint arXiv:1602.08148*, 2016.

[6] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine learning*, 56:89–113, 2004.

[7] Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012.

[8] Sebastian Böcker, Sebastian Briesemeister, Quang Bao Anh Bui, and Anke Truß. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009.

[9] Csaba Biro, Edouard Bonnet, Daniel Marx, Miltzow Tillmann, and Pawel Rzazewski. Fine-grained complexity of coloring unit disks and balls. In *33rd International Symposium on Computational Geometry (SoCG 2017)*, 2017.

[10] Sebastian Böcker and Peter Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011.

[11] Hans L Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Automata, Languages and Programming: 15th International Colloquium Tampere, Finland, July 11–15, 1988 Proceedings 15*, pages 105–118. Springer, 1988.

[12] Hans L Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 226–234, 1993.

[13] Hans L Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1-2):1–45, 1998.

[14] Hans L Bodlaender, Paul Bonsma, and Daniel Lokshtanov. The fine details of fast dynamic programming over tree decompositions. In *Parameterized and Exact Computation: 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers 8*, pages 41–53. Springer, 2013.

[15] Flavia Bonomo, Guillermo Duran, and Mario Valencia-Pabon. Complexity of the cluster deletion problem on subclasses of chordal graphs. *Theoretical Computer Science*, 600:59–69, 2015.

[16] Heinz Breu. *Algorithmic aspects of constrained unit disk graphs*. PhD thesis, University of British Columbia, 1996.

[17] Heinz Breu and David G Kirkpatrick. Unit disk graph recognition is np-hard. *Computational Geometry*, 9(1-2):3–24, 1998.

[18] Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

[19] Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64:152–169, 2012.

[20] Yixin Cao and Yuping Ke. Improved kernels for edge modification problems. *arXiv preprint arXiv:2104.14510*, 2021.

[21] Jianer Chen and Jie Meng. A 2k kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012.

[22] Brent N Clark, Charles J Colbourn, and David S Johnson. Unit disk graphs. *Discrete mathematics*, 86(1-3):165–177, 1990.

[23] Stephen A Cook. The complexity of theorem-proving procedures, stoc'71: Proceedings of the third annual acm symposium on theory of computing, 1971.

[24] Bruno Courcelle. The monadic second-order logic of graphs iii: Tree-decompositions, minors and complexity issues. *RAIRO-Theoretical Informatics and Applications*, 26(3):257–286, 1992.

[25] Biro Csaba, Edouard Bonnet, Daniel Marx, Miltzow Tillmann, and Pawel Rzazewski. Fine-grained complexity of coloring unit disks and balls. *Journal of Computational Geometry*, 2018.

[26] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Joham MM van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 150–159. IEEE, 2011.

[27] Peter Damaschke. Bounded-degree techniques accelerate some parameterized graph algorithms. In *Parameterized and Exact Computation: 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers 4*, pages 98–109. Springer, 2009.

[28] Mark De Berg, Hans L Bodlaender, Sandor Kisfaludi-Bak, Daniel Marx, and Tom C Van Der Zanden. A framework for exponential-time-hypothesis–tight algorithms and lower bounds in geometric intersection graphs. *SIAM Journal on Computing*, 49(6):1291–1331, 2020.

[29] Anders Dessmark, Jesper Jansson, Andrzej Lingas, Eva-Marta Lundell, and Mia Persson. On the approximability of maximum and minimum edge clique partition problems. *International Journal of Foundations of Computer Science*, 18(02):217–226, 2007.

[30] Absalom E Ezugwu, Abiodun M Ikotun, Olaide O Oyelade, Laith Abualigah, Jeffery O Agushaka, Christopher I Eke, and Andronicus A Akinyelu. A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future

research prospects. *Engineering Applications of Artificial Intelligence*, 110:104743, 2022.

[31] Fedor V Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Subexponential fixed-parameter tractability of cluster editing. *arXiv preprint arXiv:1112.4419*, 2011.

[32] Fedor V Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Finding, hitting and packing cycles in subexponential time on unit disk graphs. *Discrete & Computational Geometry*, 62:879–911, 2019.

[33] Fedor V Fomin, Daniel Lokshtanov, and Saket Saurabh. Bidimensionality and geometric graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1563–1575. SIAM, 2012.

[34] Fedor V Fomin and Dimitrios M Thilikos. New upper bounds on the decomposability of planar graphs. *Journal of Graph Theory*, 51(1):53–81, 2006.

[35] Yong Gao, Donovan R Hare, and James Nastos. The cluster deletion problem for cographs. *Discrete Mathematics*, 313(23):2763–2771, 2013.

[36] Michael R Garey and David S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[37] Oded Goldreich. Computational complexity: a conceptual perspective. *ACM Sigact News*, 39(3):35–39, 2008.

[38] Petr A Golovach, Pinar Heggernes, Athanasios L Konstantinidis, Paloma T Lima, and Charis Papadopoulos. Parameterized aspects of strong subgraph closure. *Algorithmica*, 82(7):2006–2038, 2020.

[39] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39:321–347, 2004.

[40] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38:373–392, 2005.

[41] Rajarshi Gupta, Jean Walrand, and Olivier Goldschmidt. Maximal cliques in unit disk graphs: Polynomial approximation. In *Proceedings INOC*, volume 2005. Citeseer, 2005.

[42] Navid Imani. Parameterized tractability and kernelization of problems on unit disk graphs. 2013.

[43] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[44] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[45] Alon Itai, Christos H Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[46] Giuseppe F Italiano, Athanasios L Konstantinidis, and Charis Papadopoulos. Structural parameterization of cluster deletion. In *WALCOM: Algorithms and Computation: 17th International Conference and Workshops, WALCOM 2023, Hsinchu, Taiwan, March 22–24, 2023, Proceedings*, pages 371–383. Springer, 2023.

[47] Hiro Ito and Masakazu Kadoshita. Tractability and intractability of problems on unit disk graphs parameterized by domain area. In *Proceedings of the 9th International Symposium on Operations Research and Its Applications (ISORA10)*, pages 120–127, 2010.

[48] Αθανάσιος Λ Κωνσταντινίδης. Algorithms and complexity of graph modification problems. 2021.

[49] Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012.

[50] Athanasios L Konstantinidis and Charis Papadopoulos. Maximizing the strong triadic closure in split graphs and proper interval graphs. *Discrete Applied Mathematics*, 285:79–95, 2020.

[51] Athanasios L Konstantinidis and Charis Papadopoulos. Cluster deletion on interval graphs and split related graphs. *Algorithmica*, 83(7):2018–2046, 2021.

[52] Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–192. IEEE, 2022.

[53] Tuukka Korhonen and Daniel Lokshtanov. An improved parameterized algorithm for treewidth. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 528–541, 2023.

[54] Mirko Křivánek and Jaroslav Morávek. Np-hard problems in hierarchical-tree clustering. *Acta informatica*, 23:311–323, 1986.

[55] David Lichtenstein. Planar formulae and their uses. *SIAM journal on computing*, 11(2):329–343, 1982.

[56] Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[57] Daniel Lokshtanov, Daniel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 777–789. SIAM, 2011.

[58] Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Jie Xue, and Meirav Zehavi. Subexponential parameterized algorithms on disk graphs (extended abstract). In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2005–2031. SIAM, 2022.

[59] Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Jie Xue, and Meirav Zehavi. A framework for approximation schemes on disk graphs. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2228–2241. SIAM, 2023.

[60] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. *Theoretical Computer Science*, 442:13–21, 2012.

[61] Sabrine Malek and Wady Naanaa. A new approximate cluster deletion algorithm for diamond-free graphs. *Journal of Combinatorial Optimization*, 39(2):385–411, 2020.

[62] Daniel Marx. The square root phenomenon in planar graphs. In *ICALP (2)*, page 28, 2013.

[63] Vijay Raghavan and Jeremy Spinrad. Robust algorithms for restricted domains. *Journal of algorithms*, 48(1):160–172, 2003.

[64] Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

[65] Mikkel Thorup. All structured programs have small tree width and good register allocation. *Information and Computation*, 142(2):159–181, 1998.

[66] Simon Tippenhauer and Wolfgang Muzler. On planar 3-sat and its variants. *Fachbereich Mathematik und Informatik der Freien Universitat Berlin*, 2016.

[67] Johan MM Van Rooij, Hans L Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *ESA*, volume 5757, pages 566–577. Springer, 2009.

[68] Johan MM van Rooij, Hans L Bodlaender, Erik Jan van Leeuwen, Peter Rossmanith, and Martin Vatshelle. Fast dynamic programming on graph decompositions. *arXiv preprint arXiv:1806.01667*, 2018.

[69] G Yu, O Goldschmidt, and H Chen. Clique, independent set and vertex cover problems in geometric graphs, 1993.