Technische Universität Berlin

Parametrised Algorithms for Finding Triangles in Graphs

Detection, Counting and Enumeration

Masterarbeit

von Matthias Bentert

zur Erlangung des Grades „Master of Science“
(M. Sc.) im Studiengang Informatik

Betreuer:
Till Fluschnik
Dr. André Nichterlein
Prof. Dr. Rolf Niedermeier

Erstgutachter: Prof. Dr. Rolf Niedermeier
Zweitgutachter: Prof. Dr. Stephan Kreutzer

Berlin, 18. November 2016

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 18. November 2016

Abstract

Detecting, counting and enumerating triangles in graphs is a simple yet frequently required task. It is well known to be solvable in $O(n \cdot m)$ time on graphs with n vertices and m edges. This thesis contributes to the relatively young field of *FPT in P* which is motivated by the question which parametrisations can be used to solve problems that are already known to be solvable in polynomial time faster in relevant special cases. We will give both positive and negative answers to this question regarding problems of finding triangles. On the one hand, we construct algorithms and kernelisations for finding triangles in graphs having constantly bounded parameter values in linear time for certain parametrisations. On the other hand, we introduce a new notion of hardness which we use to prove that certain parametrisations cannot be used to design algorithms that are faster than the best known unparametrised algorithms. Among other things, we will show that all triangles in a graph can be enumerated in $O(n \cdot k^2)$ time, where k is the degeneracy of the graph. Moreover, we will prove that counting all triangles in a graph cannot be done faster on three-colourable graphs or graphs with maximum diameter of components at least two than on arbitrary graphs. These bounds are tight as counting all triangles on bipartite (two-colourable) graphs or graphs with maximum diameter of components one, that is cluster graphs, can be done in linear time.

Zusammenfassung

Obwohl es relativ einfach ist, alle Dreiecke in einem Graphen zu zählen, hat dieses Problem die Aufmerksamkeit einiger Forscher auf sich gezogen, da es in vielen verschiedenen Anwendungen als Teilproblem auftritt. Ein einfacher Algorithmus, der alle möglichen Kombinationen durchprobiert, braucht $O(n \cdot m)$ Zeit auf einem Graphen mit n Knoten und m Kanten. Da sich viele Forscher mit diesem Problem beschäftigt haben und kein Algorithmus gefunden wurde, der Dreiecke schneller findet, wurde stattdessen versucht, Algorithmen zu entwickeln, die auf bestimmten Graphen schneller laufen. Beispiele hierfür sind ein Algorithmus, der $O(m^{1.5})$ Zeit benötigt, und ein Algorithmus, der $O(n^{2.373})$ Zeit benötigt und auf schneller Matrixmultiplikation beruht. Der erste läuft auf dünnen Graphen und der zweite auf sehr großen und dichten Graphen schneller als in $O(n \cdot m)$ Zeit. Aus der Idee solcher Algorithmen für Spezialfälle hat sich die Idee des *FPT in P* entwickelt. Dabei wird untersucht, welche Parameter genutzt werden können, um Probleme, die schon in polynomieller Zeit gelöst werden können, in relevanten Spezialfällen noch schneller zu lösen. Diesem Leitmotiv folgend werden wir sowohl positive als auch negative Antworten geben, indem wir auf der einen Seite parametrisierte Algorithmen präsentieren und auf der anderen Seite eine neue Härtedefinition einführen, die wir nutzen, um zu beweisen, dass bestimmte Parameter nicht zu schnelleren Algorithmen führen können. Wir werden unter anderem zeigen, dass alle Dreiecke in einem Graphen in $O(n \cdot k^2)$ Zeit aufgelistet werden können, wobei k die degeneracy des Graphen ist, es aber nicht möglich ist, das Problem auf dreifärbbaren Graphen oder Graphen, deren Zusammenhangskomponenten einen Durchmesser von zwei haben, schneller zu lösen als auf beliebigen Graphen. Diese Ergebnisse sind in dem Sinne optimal, dass das Problem auf zweifärbbaren Graphen und auf Graphen, deren Zusammenhangskomponenten Durchmesser eins haben, in $O(n + m)$ Zeit lösbar wird.

Contents

1 Introduction

Detecting, counting and enumerating triangles in graphs are three of the most basic tasks in algorithmic graph theory. They attract a lot of attention as they arise in various applications [8, 20, 44, 45, 55]. Currently, even for detecting triangles there is no known linear-time algorithm and it is even conjectured that such an algorithm does not exist [1]. In the young field of *FPT in $\mathcal{P}$*, to which this thesis contributes, one studies the impact of parametrisations on polynomial-time solvable problems. A parametrisation measures a property of the input like for example the maximum number of vertices a single vertex is adjacent to or the length of the longest shortest path between two vertices in a graph. The goal then is to identify parametrisations that yield algorithms whose running times depend on both the input size and the value or size of the parametrisations such that the parameterised algorithm is less dependent on the input size compared to the fastest unparametrised algorithm. If the value or size of the parametrisation is small on the input instance, then the parametrised algorithm is faster than the unparametrised algorithm. Another important aspect of parametrised complexity, which FPT in $\mathcal{P}$ is a part of, is *kernelisation*. In kernelisation one is interested in the question whether one can construct for an arbitrary input instance a new equivalent input instance whose size is upper-bounded by some function in the value or size of the parametrisation. The main contribution of this thesis is a number of parametrised algorithms, some results on kernelisation for finding triangles and a new kind of hardness proof to show parametrised intractability in the context of FPT in $\mathcal{P}$. By intractability we mean that certain parametrisations cannot be used to construct algorithms whose running time is less dependent on the input size compared to the best known unparametrised algorithm.

Graphs are used to model complex structures like social networks, protein interaction networks, the internet, road networks and many more. One of the main features in social networks like Facebook is to suggest persons you want to become friends with. In order to suggest persons which you are likely to want to become friends with and not just arbitrary people, the operators of social networks basically want to do two things. First, they want to identify groups of people which are closely linked. How closely linked a graph is, is often measured by the *clustering coefficient* [56] which is defined as three times the number of triangles in a graph divided by the number of paths of length two. The factor of three accounts for the fact that each triangle contributes to three paths of length two. The number of paths of length two can be calculated in linear time and thus, counting the number of triangles in the graph or a subgraph as fast as possible is of practical relevance in the described setting. Second, the operators of the networks want to find persons such that if you become friends with those persons, then the number of triangles in the underlying graph increases by as much as possible. This is due to the fact, that friends of friends tend to become friends themselves [60].

Further applications for counting the number of triangles in a graph include identifying websites with similar content [20], spam detection [8], complex network analysis [33, 52] and many more [9, 27, 29, 61]. Listing all triangles in a graph is for example essential to find so-called motifs in protein interaction networks [65], which can be used to correlate the topological and functional properties of proteins [64].

FPT in $\mathcal{P}$ is a new topic in parametrised complexity analysis. In parametrised complexity, one is interested in the question whether a problem becomes "tractable" when some parameter of the input is fixed [16, 18, 26, 53]. FPT in $\mathcal{P}$ describes results on problems whose unparametrised variant can be solved in polynomial time [31]. Take as an example the maximum degree Δ, that is, the maximum number of neighbours a vertex has in the graph. To list all triangles, one does not have to check for all $\binom{n}{3}$ possibilities of three vertices but only $n \cdot \binom{\Delta}{2}$ possibilities (for each vertex and each combination of two of its at most Δ neighbours). We will investigate different problems of finding and enumerating triangles and show results on several parametrisations. On the one hand, we will show which parametrisations yield kernelisations and solving algorithms whose worst-case running time is faster than the worst-case running time of the fastest algorithms for the unparametrised versions of the problems. On the other hand, we will prove that some parametrisations cannot be used to do so.

1.1 Related Work

Since finding triangles in graphs is of great relevance, it has been studied from multiple points of view. For a list of (unparametrised) algorithms see for example Latapy [45]. For a list with further applications of finding triangles in graphs we refer to Kolountzakis et al. [44]. They also give short explanations why finding triangles is important in each of the applications. Schank and Wagner [55] implemented different algorithms to count and list all triangles in large real-world graphs. Schank and Wagner [56] presented an algorithm which runs in linear time to approximate the clustering coefficient. The clustering coefficient can be translated into the number of triangles in a graph in linear time since one can compute the number of paths of length two in linear time. Finding triangles in graphs has also been studied in the context of the streaming model [6, 11]. In the streaming model, not the whole graph is stored but instead one iterates over the input (possibly multiple times).

From a more theoretical point of view, Abboud and Williams [1] have conjectured that there is no linear-time algorithm for triangle detection. Even though this is only an unproven conjecture, it seems plausible since, despite significant efforts, no linear-time algorithm is known. Furthermore, Williams and Williams [63] proved that if deciding whether an edge-weighted graph contains a triangle of negative total edge weight in subcubic time, then other tasks like the all-pair shortest paths problem and boolean matrix multiplication can be done in subcubic time. These problems are well studied, of practical interest, can be used to solve other problems and are not known to be solvable in truly subcubic time [12, 17, 46]. In order to decide whether an edge-weighted graph contains a triangle of negative total edge weight, we will give parametrised algorithms that list all triangles in a graph. These algorithms can be used to decide whether there is a triangle with negative total edge weight in the graph without altering the asymptotic worst-case running time.

Triangles are arguably one of the most basic structures in graphs except for vertices and edges. Two very natural generalisations of triangles are cliques (complete subgraphs) and cycles. Finding maximum-cardinality cliques in graphs is of great practical importance [28], but they are $\mathcal{NP}$-hard to compute [51] and even hard to approximate [21]. Eppstein et al. [23] presented an algorithm to list all maximal cliques in a graph. Finding and counting minimum-length cycles in graphs, on the other hand, were studied by Itai and Rodeh [40] and Alon et al. [4], respectively.

This thesis follows the FPT in $\mathcal{P}$ approach presented by Giannopoulou et al. [31]. The main motivation behind this approach is to get more efficient algorithms for problems with polynomial running times, such as finding triangles, by constructing fixed-parameter algorithms. Mertzios et al. [49] showed results in the field of FPT in $\mathcal{P}$ for finding a maximum-cardinality matching. Abboud et al. [2] designed fixed-parameter algorithms to compute the diameter and the radius of a graph. Green and Bader [34] used vertex cover sets as a parametrisation to count the number of triangles in a graph. Their experimental study showed that their algorithm is ten to thirty percent faster than the best known unparametrised algorithms on test data provided by Bader et al. [5]. This result is to our knowledge the closest to this thesis and should be seen as motivation to dive further into the topic.

1.2 Results

Our results can be divided into three categories. First, we design fast parametrised algorithms to find triangles. Second, we prove that finding triangles admits kernelisations with several parametrisations. A kernelisation is a collection of polynomial-time preprocessing steps that reduce the size of the input such that it is upper-bounded by some function of the parameter size (or value). Third, we introduce a new notion of hardness and show how this can be used to prove that finding triangles cannot be solved significantly faster with certain parametrisations than the unparametrised version.

We want to especially highlight the following results. Theorem 3.8 states that Triangle Enumeration parametrised by degeneracy d is solvable in $O(n \cdot d^2)$ time. Degeneracy is a well-motivated parametrisation [32] and thus this result is well-suited for practical usage. Theorem 3.25 states that Triangle Enumeration parametrised by maximum degree Δ and deletion set to d-degenerate graphs of size k is solvable in $O(n \cdot d^2 + k \cdot \Delta^2)$ time. Green and Bader [34] showed that Triangle Counting can be solved in $O(k \cdot \Delta^2)$ where k is the size of a vertex cover. Note, that a vertex cover is a deletion set to 0-degenerate graphs and hence for every fixed d we use weaker (smaller) parametrisations. Note further, that our algorithm lists all triangles instead of only counting them. Summarizing, our result generalises the result by Green and Bader, in the sense that we use a more general parameter to solve a computational harder problem in essentially the same worst-case running time.

Preprocessing and kernelisation are powerful tools in parametrised complexity but most often used for problems for which no polynomial-time algorithm is known. Theorem 4.4 shows that for Vertex-Weighted Triangle Counting there is a preprocessing step that takes linear time and yields an exponential-size kernel for different parametrisations. The weighted version is a natural extension of Triangle

COUNTING and using it combined with the described kernel can be used to solve TRIANGLE COUNTING faster.

Lastly, Theorem 5.7 proves that even when restricting TRIANGLE DETECTION to graphs with a minimum dominating set of size three, the worst-case time complexity remains unchanged.

A list of all our results on algorithms solving TRIANGLE DETECTION, TRIANGLE COUNTING, TRIANGLE ENUMERATION, GIRTH or c-CLIQUE and their running times is shown in Table 1. Table 2 gives an overview over all our kernelisation results. Moreover, we prove that TRIANGLE DETECTION cannot be solved any faster when parametrised by minimum degree, bisection-width, maximum diameter of components, minimum dominating set or chromatic number than the unparametrised version.

Using a "parameter hierarchy"[57], Figure 1 shows a condensed form of all our results on TRIANGLE DETECTION. In a parameter hierarchy a connection between two parametrisaitons A and B, where A is above B, indicates that A upper bounds B. That is, there is some function f such that in every possible input instance I, the value (or size) of B in I is at most $f(a)$, where a is the value (or size) of A in I. We coloured the parametrisation by the running times the respective algorithms achieve. Running times independent of the input size are coloured gray, running times linear in the number of vertices are coloured green or darkgreen, running times linear in the number of vertices and edges are coloured yellow or darkyellow and running times quadratic in the number of vertices are coloured orange. If a parametrisation cannot be used to design fast parametrised algorithms, we coloured it red or darkred. A darker colour indicates that this result is deduced by an other result on a smaller or larger parameter. Tractability transfers to parameters above and intractability to parameters below in the parameter hierarchy. See Section 2.4.2 for more information and an introduction to the parameter hierarchy in general.

1.3 Outline

The outline of this thesis is as follows. In Section 2 we will settle some preliminaries and give formal definitions of the different problems and various parametrisation we use in this thesis. In Section 3 we will describe parametrised algorithms to solve triangle finding problems and in Section 4 we will describe algorithms which yield kernels whose size is upper-bounded by some function of the parameter size. In Section 5 we will introduce a new kind of hardness proof and use it to show that finding triangles cannot be done faster with certain parametrisations than it can be without them. Lastly, in Section 6 we give a short summary of this thesis and present some directions for future work.

Table 1: Our results on solving algorithms for the different problems we study. The function f is some computable function, n, m and #T are the number of vertices, edges and triangles in the input graph. The size (or value) of the parameter is k and $\omega < 2.373$ is the fast matrix multiplication exponent. The maximum degree is denoted by Δ. The number c is part of the input of c-Clique and the question is whether there is a c-clique in the graph.

parametrisation	running time	reference
	Triangle Detection	
clique-width	$f(k) \cdot (n+m)$	Corollary 3.12
deletion set to chordal graphs	$O(n \cdot k^2)$	Proposition 3.14
deletion set to perfect graphs	$O(n + m \cdot k)$	Proposition 3.16
distance to clique	$O(1)$	Observation 3.1
maximum independent set size	$O(k^{2\omega})$	Corollary 3.5
	Triangle Counting	
deletion set to chordal graphs	$O(n + m \cdot k)$	Observation 3.23
h-index	$O(n + m \cdot k)$	Proposition 3.7
	Triangle Enumeration	
average degree	$O(n^{1.5} \cdot k^{1.5})$	Proposition 3.10
clique-width	$O(n^2 \cdot k^2 + \#\mathrm{T})$	Theorem 3.13
degeneracy	$O(n \cdot k^2)$	Theorem 3.8
deletion set to bipartite graphs	$O(n + m \cdot k)$	Corollary 3.20
deletion set to chordal graphs	$O(n + m \cdot k + \#\mathrm{T})$	Theorem 3.22
deletion set to cographs	$O(n + m \cdot k + \#\mathrm{T})$	Theorem 3.22
feedback edge number	$O(k^2 + n + m), O(n \cdot k + m)$	Theorem 3.6
deletion set to d-degenerate graphs and maximum degree	$O(n \cdot d^2 + k \cdot \Delta^2)$	Theorem 3.25
	Girth	
deletion set to perfect graphs	$O(n^2 + k \cdot (n+m))$	Proposition 3.17
distance to clique	$O(k^3)$	Corollary 3.3
	c-Clique	
clique-width	$f(k) \cdot (n+m)$	Proposition 3.11
degeneracy	$O(n \cdot k^{c-1} \cdot c^2)$	Corollary 3.9
deletion set to bipartite graphs	$O(n + m \cdot k^{c-2} \cdot c^2)$	Corollary 3.21
deletion set to chordal graphs	$O(n \cdot k^{c-1} \cdot c^2 + m)$	Corollary 3.24
deletion set to perfect graphs	$O(n^{c-1} \cdot k \cdot c^2 + (n+m)^{O(1)})$	Proposition 3.18
distance to clique	$O(1)$	Corollary 3.2
deletion set to d-degenerate graphs and maximum degree	$O(n \cdot d^{c-1} \cdot c^2 + k \cdot \Delta^{c-1} \cdot c^2)$	Corollary 3.26

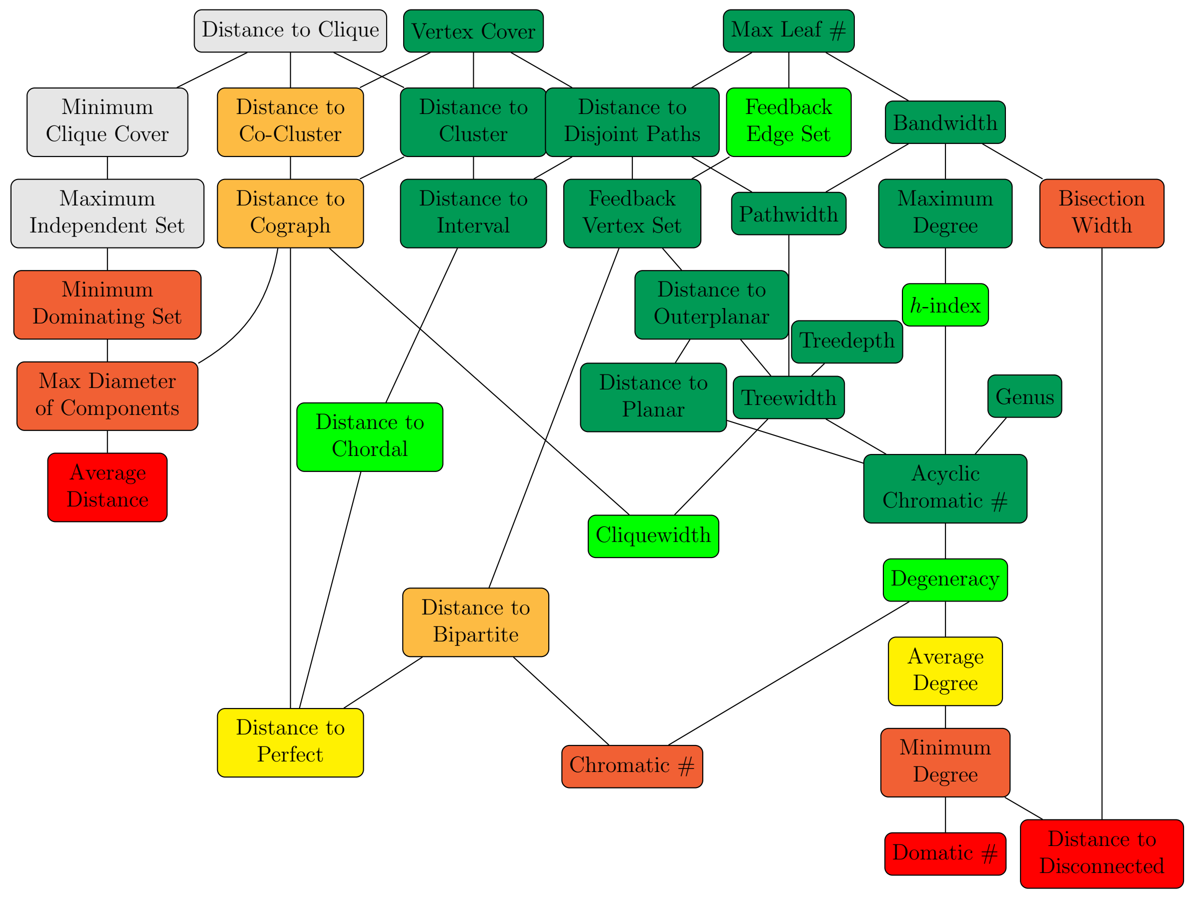

Figure 1: Parameter hierarchy and results for TRIANGLE DETECTION. The colouring is based on the running time the respective algorithms achieve.
There is an algorithm whose running time is only dependent on the parameter size (or value).
There is an algorithm whose running time is linear in the number of vertices.
There is an algorithm whose running time is quadratic in the number of vertices.
There is probably no subquadratic-time algorithm.
There is an algorithm whose running time is linear in the number of vertices. This is deduced from an other parameter.
There is an algorithm whose running time is quadratic in the number of vertices. This is deduced from an other parameter.
There is probably no subquadratic-time algorithm. This is deduced from an other parameter.

Table 2: Our results on kernelisation, where n, m are the number of vertices and edges, Δ is the maximum degree in the input graph and k is the size (or value) of the parameter. The abbreviation VWTC stands for Vertex-Weighted Triangle Counting.

parametrisation	problem	kernel size	running time
deletion set to d-degenerate graphs	VWTC	$O(2^{2k})$	$O(n \cdot d \cdot (k+d))$
feedback edge number	Counting	$O(k)$	$O(n+m)$
maximum independent set size	Detection	$O(k^4)$	$O(1)$
deletion set to d-degenerate graphs and maximum degree	Counting	$O(k^2 \cdot \Delta^2)$	$O(n \cdot d \cdot (k+d))$
vertex cover and maximum degree	Counting	$O(k \cdot \Delta)$	$O(n+m)$

2 Preliminaries and Basic Observations

In this section we will introduce concepts needed for this thesis and we will fix some notation. First, we start with general preliminaries. Next, we define the problems discussed in this thesis. Third, we provide some information on these problems. Last, we will introduce and motivate the different parametrisations we use.

2.1 General Preliminaries

Every problem in this thesis takes as input an undirected graph $G = (V, E)$. In this context, we denote by

V	the *vertex set* of G;
E	the *edge set* of G;
n	the number of vertices, formally, $n := \|V\|$;
m	the number of edges, formally, $m := \|E\|$;
$\|G\|$	the *size* of G, formally, $\|G\| := n + m$;
$N_G(v)$	the *neighbourhood* of v in G, formally, $N_G(v) := \{w \in V \mid \{v, w\} \in E\}$;
$\deg_G(v)$	the *degree* of v, formally, $\deg_G(v) := \|N_G(v)\|$;
Δ_G	the *maximum degree* in G, formally, $\Delta_G := \max_{v \in V}\{\deg_G(v)\}$;
$\#\mathrm{T}_G$	the *number of triangles* in G, formally, $\#\mathrm{T}_G := \|\{\{x, y, z\} \mid \{x, y\} \in E \land \{x, z\} \in E \land \{y, z\} \in E\}\|$;
$G[U]$	the subgraph of G induced by $U \subseteq V$, formally, $G[U] := (U, \{\{x, y\} \in E \mid x \in U \land y \in U\})$;

```
1:  # n = 5
2:  # m = 5
3:  []
4:  [3, 4]
5:  [2, 4, 5]
6:  [2, 3, 5]
7:  [3, 4]
```

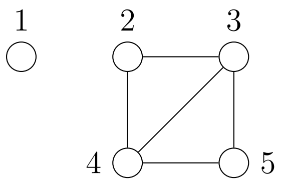

Figure 2: Example of an input graph.

$G - U$	the graph obtained from G by deleting the vertices in $U \subseteq V$, formally, $G - U := G[V \setminus U]$.

If the graph G is clear from the context, then we will omit the subscript G.

We assume the input graph G to be given by its number of vertices and edges, and the adjacency lists for all vertices. To fix an arbitrary ordering of the vertices in constant time as well as reducing the size of the input, we assume the vertices to be integers in between 1 and n. Figure 2 illustrates an example of such an input. For the sake of readability, we will not represent vertices by integers in this thesis but give them names which we consider to be more readable like for example u, v, x, y or v_1, v_2, v_3. Lastly, we assume the edges to be stored in a hash table after reading the input in order to achieve constant amortized insert and look-up time. In practice it might be better to use other representations like ordered adjacency lists for the edges which might add a logarithmic factor to the running time but achieve better constants and faster algorithms in practice.

The *parametrisation* of a problem is a concept that reflects certain properties or structures of the input. For example, the parametrisation maximum degree Δ is the maximum number of vertices a single vertex is adjacent to. The *parameter* of a graph is the actual integer, set, etc. of the respective parametrisation. The parameter maximum degree of the graph seen in Figure 2 is three. The parameter can be an integer as well as a set, an expression, a graph or anything else. However, we require the parameter to have an integer associated with it in order to use this integer in the running time analysis. When the parameter is an integer, then we denote the parameter by k and the integer associated with it by $||k|| := k$. When the parameter is a set, then we denote the parameter with K and the integer associated with it by $||K|| := |K|$. To avoid overload with the absolute value, we denote the amount of space an integer or function i requires in the memory by $\langle i \rangle$.

A *data reduction rule* is a polynomial-time preprocessing step that may reduces the size of the input. Formally, it is defined as a mapping from one input instance I of a problem to a new instance I' of the same problem with $|I'| \leq |I|$ such that either both are yes-instances or both are no-instances.

Let $\mathbb{Q}^+$ be the set of all positive rational numbers. The set $\mathcal{NP}$ is the set of all problems for which there exists a non-deterministic algorithm solving the problem in polynomial time. The fast matrix multiplication exponent is $\omega < 2.373$ [62].

2.2 Problem Description

In this subsection we give a formal definition of the problems discussed in this thesis. Moreover, we show which of these problems are special cases of each other and discuss the implications of those relationships between problems.

The first problem we discuss is TRIANGLE DETECTION. It takes an undirected graph as input and the question is whether the graph contains a triangle.

TRIANGLE DETECTION
Input: An undirected graph G.
Question: Does G contain a triangle?

Next, we give a definition for TRIANGLE COUNTING. It takes an undirected graph and a non-negative integer t as input and the question is whether the graph contains at least t triangles.

TRIANGLE COUNTING
Input: An undirected graph G and an integer $t \geq 0$.
Question: Does G contain at least t triangles?

The third problem we want to discuss is TRIANGLE ENUMERATION. It takes an undirected graph as input and the task is to list all triangles in the graph.

TRIANGLE ENUMERATION
Input: An undirected graph G.
Task: List all triangles in G.

The fourth problem is c-CLIQUE. It takes an undirected graph and a positive integer c as input and the question is whether the graph contains a c-clique, that is a subgraph of c vertices such that all of these vertices are pair-wise adjacent. Note that this definition is basically the same as a definition that asks for a clique of size at least c, since every clique of size greater than c contains cliques of size c as subgraphs.

c-CLIQUE
Input: An undirected graph G and an integer $c \geq 1$.
Question: Does G contain a clique of size c?

Next, we discuss GIRTH. It takes an undirected graph and an integer $g \geq 3$ as input and the question is whether the graph contains a cycle of length at most g. A cycle of length g is a sequence of vertices, such that no vertex appears twice in the sequence, each two consecutive vertices in the sequence are adjacent to each other in the graph and the first and the last vertex in the sequence are also adjacent in the graph.

GIRTH
Input: An undirected graph G and an integer $g \geq 3$.
Question: Does G contain a cycle of length at most g?

Last, we give a definition for Vertex-Weighted Triangle Counting. It takes an undirected graph, a positive rational number t and a function w which assigns each vertex a positive rational weight as input. The weight of a triangle $\{x, y, z\}$ is defined as $w(x) \cdot w(y) \cdot w(z)$ and the question is whether the sum of the weights over all triangles is at least t.

Vertex-Weighted Triangle Counting
Input: An undirected graph G, a number $0 < t \in \mathbb{Q}^+$ and a weight function $w \colon V \to \mathbb{Q}^+$.
Question: Is the sum of the weights $w(x) \cdot w(y) \cdot w(z)$ over all triangles $\{x, y, z\}$ at least t?

We will show that all of the described problems are closely related. We will prove that certain problems are special cases of others, by proving that any algorithm which solves one problem in $O(x)$ time can be used to solve the special case in $O(x)$ time as well.

First we will show that Triangle Detection is a special case of Triangle Counting. This immediately follows from the observation that a graph contains a triangle if and only if the number of triangles the graph contains is at least one.

Observation 2.1. *Every algorithm that solves* Triangle Counting *in $O(x)$ time can be used to solve* Triangle Detection *in time $O(x)$ time.*

Second, we will show that Triangle Counting is a special case of Triangle Enumeration. Listing all triangles in G and then count the number of triangles listed solves Triangle Counting.

Proposition 2.2. *Every algorithm that solves* Triangle Enumeration *in $O(x)$ time can be used to solve* Triangle Counting *in $O(x)$ time.*

Proof. Let $\mathcal{A}$ be an algorithm that solves Triangle Enumeration in $O(x)$ time and let #T be the number of triangles in G. Since $\mathcal{A}$ lists all triangles in G, it holds that $\#\mathrm{T} \in O(x)$. Therefore an algorithm $\mathcal{B}$ that takes the output of $\mathcal{A}$ as input and counts the number of triangles that $\mathcal{A}$ produces has running time in $O(x)$. First applying $\mathcal{A}$, then $\mathcal{B}$ and then check whether the output of $\mathcal{B}$ is at least t solves Triangle Counting in $O(x)$ time. $\square$

Next, we will prove that Triangle Counting is a special case of Vertex-Weighted Triangle Counting.

Proposition 2.3. *Every algorithm that solves* Vertex-Weighted Triangle Counting *in $O(x)$ time can be used to solve* Triangle Counting *in $O(x)$ time.*

Proof. Let $\mathcal{A}$ be an algorithm that solves Vertex-Weighted Triangle Counting in $O(x)$ time. Applying $\mathcal{A}$ on (G, t, w) where for any $v \in V$ it holds that $w(v) = 1$, solves Triangle Counting in $O(x)$ time. This construction is correct since every triangle has weight one and thus the sum over the weights of all triangles equals the number of triangles in G. $\square$

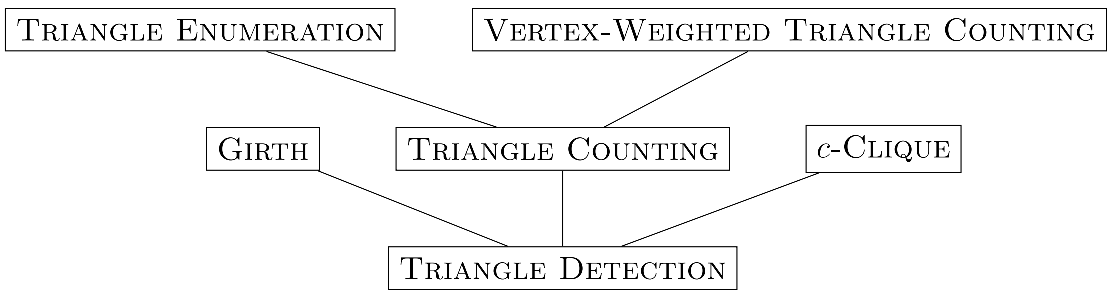

Figure 3: Hasse diagram of the described problems.

TRIANGLE DETECTION is a special case of GIRTH and c-CLIQUE. A graph contains a triangle if and only if it contains a cycle of length three.

Proposition 2.4. *Every algorithm that solves* GIRTH *in $O(x)$ time can be used to solve* TRIANGLE DETECTION *in $O(x)$ time.*

Proof. Let $\mathcal{A}$ be an algorithm that solves GIRTH in $O(x)$ time. Applying $\mathcal{A}$ on $(G, 3)$ solves TRIANGLE DETECTION in $O(x)$ time, since a cycle of size three is a triangle. □

A graph contains a triangle if and only if it contains a clique of size three.

Observation 2.5. *Every algorithm that solves c-*CLIQUE *in $O(x)$ time can be used to solve* TRIANGLE DETECTION *in $O(x)$ time.*

Proof. Let $\mathcal{A}$ be an algorithm that solves c-CLIQUE in $O(x)$ time. Applying $\mathcal{A}$ on $(G, 3)$ solves TRIANGLE DETECTION in $O(x)$ time, since a clique of size three is a triangle. □

Summarizing, Figure 3 shows the relationship between the described problems. An edge between two problems illustrates that the lower one is a special case of the upper one.

We now discuss the implications of such relations between problems. Let X be the special case of a problem Y. We have already shown that if Y can be solved in $O(x)$ time for some x, then X can be solved in $O(x)$ time, too. It follows by contraposition, that if X cannot be solved in $O(x)$ time, then neither can Y. We will omit proofs which follow directly from this simple observation.

2.3 Reduction Rules and Basic Observations

We will provide basic information and some reduction rules for the various problems defined above. Let $2 \leq \omega < 2.373$ be the fast matrix multiplication exponent. TRIANGLE COUNTING can be solved in $O(n^\omega)$ time by the following algorithm. First compute the adjacency matrix A of G in $O(n^2)$ time. Compute the matrix $B = A^3$ in $O(n^\omega)$ time. Sum up all values on the diagonal of B and divide this value by six. This computed integer is the number of triangles in G. For more information see for example Latapy [45].

Triangle Enumeration can be solved in $O(n^3)$ time by trying all possible combinations of three vertices and and checking in constant time whether these three vertices build a triangle and if so, then list this triangle. Note that a complete graph with n vertices contains $n(n-1)(n-2)/6$ triangles and it therefore takes $O(n^3)$ time to list all triangles. For more information, a formal proof and an algorithm listing all triangles in $O(n^3)$ time see Latapy [45].

The problem c-Clique is known to be $W[1]$-hard [18] when parametrised by c. This means that it is assumed very unlikely that there is an $f(c) \cdot (n+m)^{O(1)}$-time algorithm solving c-Clique. Moreover Nešetřil and Poljak [51] showed that the brute force algorithm that takes $O(n^c)$ time is basically the best we can achieve with unparametrised complexity analysis.

Itai and Rodeh [40] showed that Girth can be solved in $O(n \cdot (n+m))$ time and that if g is even, then Girth can be solved in $O(n^2)$ time.

Vertex-Weighted Triangle Counting can be solved in $O(n^\omega)$ time by the same algorithm which solves Triangle Counting. Note that the adjacency matrix A has to contain the weights of the edges but only the vertices are weighted. One can easily overcome this gap by defining $A_{i,j} = \sqrt{w(i) \cdot w(j)}$. This is correct since every triangle $\{x, y, z\}$ has weight

$$w(x) \cdot w(y) \cdot w(z) = \sqrt{w(x) \cdot w(y)} \cdot \sqrt{w(x) \cdot w(z)} \cdot \sqrt{w(y) \cdot w(z)}.$$

We now present reduction rules for the different problems and prove their correctness.

The first reduction rule is based on Turán's theorem, which states that every graph which does not contain a complete subgraph K_{c+1} of $c+1$ vertices contains at most $\frac{n^2}{2} \cdot (1 - \frac{1}{c})$ edges.

Thus, an instance of c-Clique can be solved immediately, if $m > (n^2 \cdot (1 - 1/c))/2$.

Reduction Rule 2.1 (Aigner and Ziegler [3])**.** *If $m > \frac{n^2}{2} \cdot (1 - \frac{1}{c})$, then return a trivial yes-instance.*

Moreover, an instance of Triangle Detection can be solved immediately, if $m > n^2/4$.

Reduction Rule 2.2. *If $m > \frac{n^2}{4}$, then return a trivial yes-instance.*

Proof. A triangle is a K_3, that is a complete subgraph of three vertices. Thus, if G does not contain a triangle, then it contains at most $\frac{n^2}{2} \cdot (1 - \frac{1}{2}) = \frac{n^2}{4}$ edges. □

The next reduction rule is based on the same idea but uses the maximum degree to achieve a tighter bound on the number of edges.

Reduction Rule 2.3. *If $\Delta > \frac{n}{2}$ and $m > (n - \Delta) \cdot \Delta$, return a trivial yes-instance.*

Proof. We will show that every triangle-free graph with n vertices and maximum degree $\Delta \geq \frac{n}{2}$ contains at most $(n - \Delta) \cdot \Delta$ edges.

Let G be an arbitrary triangle-free graph with n vertices and maximum degree Δ. Since the maximum degree is Δ, there is a vertex v with $\deg(v) = \Delta$. Note that G is triangle-free and each edge between two neighbours u, w of v would imply a triangle $\{u, v, w\}$. Thus, $N(v)$ is an independent set and hence every vertex in $N(v)$ has at most $|V \setminus N(v)| = n - \Delta$ neighbours. Since Δ is the maximum degree in G, each vertex in $V \setminus N(v)$ has at most Δ neighbours. Therefore, G contains at most $(\Delta \cdot (n - \Delta) + (n - \Delta) \cdot \Delta)/2 = (n - \Delta) \cdot \Delta$ edges. □

Consider the full bipartite graph $G' = K_{n-\Delta,\Delta}$ with $n - \Delta$ vertices in one part U of the partition and Δ vertices in the other part W. By construction, G' is triangle-free, has maximum degree $\Delta > \frac{n}{2} \geq n - \Delta$ and contains an edge for each pair of vertices u, w with $u \in U$ and $w \in W$. Hence, G' contains $|U| \cdot |W| = (n - \Delta) \cdot \Delta$ edges and thus the given bound is tight.

The following reduction rule for Triangle Counting is based on a result by Itai and Rodeh [40].

Reduction Rule 2.4. *If $t > 2m^{1.5}$, then return a trivial no-instance.*

Proof. Consider the following algorithm:

Find a rooted spanning tree T for each connected component C with at least three vertices. Let E_T be the edges in T, $E' = E_C \setminus E_T$ the edges in C not in T and $p(v)$ denote the parent of v in T. For each edge $e = \{u, v\} \in E'$ check whether $\{u, p(v)\} \in E'$ or $\{p(u), v\} \in E'$. If one of these conditions hold or if $p(u) = p(v)$ reduce t by one, if both hold and $p(u) \neq p(v)$ reduce t by two. Delete E_T from E and repeat until no connected component has at least three vertices.

This algorithm counts all triangles in G and counts at most $2m^{1.5}$ triangles, because each iteration can count at most $2m$ triangles and there are at most $\sqrt{m}$ many iterations as shown by Itai and Rodeh [40]. □

The following reduction rule can be applied to every problem discussed in this thesis except for c-Clique when $c < 3$. The reduction rule is based on the observation that every vertex in a triangle or cycle has at least two neighbours.

Reduction Rule 2.5. *Delete all vertices with degree* 0 *or* 1*.*

Proof. Since every vertex in a triangle or cycle has at least degree two, vertices with degree zero or one cannot be part of a triangle. □

2.4 Parametrisations

We will introduce all the parametrisations we use in this thesis. First, we will provide information on how these parametrisations are defined and how fast one can compute them. Moreover, we will introduce different concepts associated with those parametrisations and we will analyse which parametrisations are well suited for finding triangles. Keep in mind that each of the problems we introduced might be a subroutine of a more complex problem and thus even a parametrisation which is $\mathcal{NP}$-hard to compute might be already computed prior to the finding triangle procedure. Nonetheless, a parametrisation which can be computed in linear time and which yields an algorithm as fast as an algorithm using a parametrisation which is hard to compute is in general more beneficial. Second, we will have a look at the parameter hierarchy, that is, a network of relations between different parametrisations.

2.4.1 Characterisations

We now give formal definitions for the different parametrisations we use in this thesis. The selection of the parametrisations is based on Sorge and Weller [57]. We ordered the parametrisations such that related definitions come directly after another and that the complexity of the definitions of the parametrisations increase over the course of this section. Some of the parameters are usually quite large and hence using them for $\mathcal{NP}$-hard problems where they affect the running time by an exponential amount often yield algorithms that are to slow for practical usage. However, we only consider problems that require polynomial time to solve and most of these parameters are smaller than n. Hence, the parameters may only affect the running time by a polynomial amount and they may still yield algorithms that are faster than the best known unparametrised algorithms (and thus fast enough for practical usage).

Minimum degree The *minimum degree* k in a graph is the minimum size of a neighbourhood of one vertex, that is $k = \min\{|N(v)| \mid v \in V\}$. Although this parameter can be computed in linear time, we will show in this thesis that it can not be used to construct faster algorithms for any of the defined problems.

Average degree The *average degree* $k = 2m/n$ is a measurement of how sparse the graph is. If n and m can be read in constant time, then k can also be computed in constant time. Otherwise it takes linear time to read the whole input and compute n, m and k. Since in many real-world applications the graphs are sparse, it is worth a shot to compute the average degree and if it is low, then use it to compute triangles fast.

Maximum degree The *maximum degree* Δ is the maximum size of a neighbourhood of a vertex. Formally this is $\Delta = \max\{|N(v)| \mid v \in V\}$. This parameter can be computed in linear time and since it is often much smaller than n, it seems promising to use. However, the degeneracy of a graph can often be used in the same way and is at most as large as the maximum degree. We will however use the maximum degree of subgraphs in this thesis.

h-index The *h-index* is the maximum number k such that there are at least k vertices having degree at least k. This parameter can be computed in linear time and is always smaller than the maximum degree. If the graph has rather few vertices with many neighbours, then this parametrisation might be fitting. Since the degree distribution of many real-world graphs such as social networks or the world wide web follows a power law [25, 43], the h-index of those graphs is small compared to the size of the graph.

Degeneracy The *degeneracy* of a graph G is the smallest number k such that G and every subgraph of G contain a vertex of degree at most k. A *degeneracy order* of G is an order $\leq_d$ such that for each vertex $v \in V$ it holds that $|N(v) \cap \{w \mid v \leq_d w\}| \leq k$. That is, every vertex has at most k neighbours which come later in the degeneracy order. Closely related to the degeneracy order is the degeneracy ordering, that is, an ordering $v_1, \ldots, v_n$ of the vertices such that it holds $v_i \leq_d v_{i+1}, i \in [0, n-1]$. Batagelj and Zaversnik [7] showed how to compute the degeneracy and the degeneracy ordering in linear time. One can use this ordering by iterating over each vertex v in this order and compute all triangles which have v as their first vertex in this order. Therefore one has to only check at most k neighbours for each vertex. This parametrisation is very well motivated for two reasons. First, many other well-known parametrisations upper-bound the degeneracy and thus a fast algorithm using the degeneracy can be used to construct fast algorithms using these other parametrisations. Second, the degeneracy of many real-world applications such as for example social networks analysis is small compared to the size of the graph since the degree distribution of graphs in those applications follows a power law.

Maximum diameter of components The *maximum diameter of components* is the maximum number k such that there are two vertices u, v such that the shortest path between those two is of length k. It can be computed in $O(n \cdot m)$ time. Since TRIANGLE ENUMERATION can be solved in $O(n \cdot m)$ time, this parametrisation is in general not suited to design faster algorithms for TRIANGLE ENUMERATION. Moreover, we will show in this thesis that even when the maximum diameter of components is given, there is in general no way to use this to solve TRIANGLE DETECTION faster than without this information.

Vertex cover A *vertex cover* of a graph G is a set K of vertices such that each edge has at least one endpoint in K. Equivalently, the graph $G' = G - K$ only contains isolated vertices. Computing a minimum vertex cover is $\mathcal{NP}$-hard [42] but there are good heuristics and even a 2-approximation for vertex cover that run in linear time. Since a vertex cover of a graph is usually quite large, this parametrisation seems unfitting at first. However, the size of a vertex cover is smaller than n and due to this Green and Bader [34] were able to implement an algorithm based on vertex cover. They showed that this algorithm is about ten to thirty percent faster than the unparametrised algorithm on some experimental data they used.

Feedback edge set A *feedback edge set* is a set K of edges such that the graph $G' = (V, E \setminus K)$ is a forest, that is, a cycle-free graph. The *feedback edge number* is the size of a minimum feedback edge set. Both the feedback edge number and a minimum-size feedback edge set can be computed in linear time using depth-first search as seen in Algorithm 1. The set K contains a minimum feedback edge set after the algorithm terminates and $|K|$ is the feedback edge number.

Algorithm 1 Feedback edge set in $O(n + m)$ time.

```
 1: let $S$ be an empty stack;
 2: $D := \emptyset$;
 3: $K := \emptyset$;
 4: while $|D| \neq n$ do
 5:     pick an arbitrary vertex $u$ from $V \setminus D$;
 6:     $S$.push($u$);
 7:     $D := D \cup \{u\}$;
 8:     while $S$ is not empty do
 9:         $v := S$.pop();
10:         for $w \in N(v)$ do
11:             if $w \in D$ then
12:                 $K := K \cup \{\{v, w\}\}$;
13:             else
14:                 $S$.push($w$);
15:                 $D := D \cup \{w\}$;
```

Bisection width The *bisection width* of a graph is the minimum number k such that there is a set K of edges with $|K| = k$ such that the vertices in the graph $G' = (V, E \setminus K)$ can be partitioned into two sets A, B whose sizes differ by at most one and there is no edge between vertices in A and vertices in B. Computing such a set K is $\mathcal{NP}$-hard [30] and we will show that even when the set is given, finding triangles cannot be done significantly faster.

Chromatic number The *chromatic number* of a graph is the minimum number of different labels one needs in order to assign each vertex exactly one label such that no edge has two endpoints with the same label. Computing the chromatic number is $\mathcal{NP}$-hard [42]. We will show that even when an optimal labeling of the graph is given, finding triangles cannot be done faster.

Minimum dominating set A *minimum dominating set* is a minimum set K of vertices such that each vertex $v \in V$ is in K or has at least one neighbour in K. Computing a minimum dominating set is $\mathcal{NP}$-hard [42] and we will prove that even when such a set K is given, finding triangles is still as hard as in the unparametrised case.

Maximum independent set A *maximum independent set* is a maximum set K of vertices such that each two vertices in K are non-adjacent. Computing a maximum independent set is $\mathcal{NP}$-hard [58]. Unfortunately, all our results on finding triangles parametrised by an independent set require a maximum independent set and not just any set of vertices whose vertices are pairwise non-adjacent. Using this parametrisation therefore seems not promising, unless it is already computed for other purposes.

Clique-width The *clique-width* of a graph G is the minimum number k such that G can be constructed using a k-expression. A k-expression consists of four operations which use k labels. The operations are the following.

- Creating a new vertex with some label i.
- Disjoint union of two labeled graphs.
- Edge insertion between every vertex with label i to every vertex with label j for some labels $i \neq j$.
- Renaming of label i to j for some i, j.

Computing the clique-width of a graph is $\mathcal{NP}$-hard [59], but there are heuristics to compute the optimum k-expressions using fast SAT-solvers [38]. We currently do not have any information on how large the clique-width of graphs in real-world applications is.

Next we will describe several parametrisations which measure the distance to a prespecified graph class Π. A *deletion set to* Π is a set K of vertices in the graph G such that $G' = G - K$ is a graph in Π. The *distance to* Π is the smallest number k such that there is a deletion set of size k to Π.

Clique A *clique* is a graph in which any two vertices are pairwise adjacent. Such graphs can be recognised in linear time by computing n and m and checking whether $m = \binom{n}{2}$. Note that this takes constant time if n and m can be computed in constant time. It is however $\mathcal{NP}$-hard to compute a clique C of maximum size in a graph [42]. Since a triangle is a clique of size three, it seems unpromising to compute the distance to a larger clique in order to solve any of the problems defined above.

Feedback vertex set A *feedback vertex set* is a set K of vertices such that $G' = G - K$ is a forest. Since every forest is 1-degenerate and every 1-degenerate graph is a forest, feedback vertex set can be seen as a deletion set to 1-degenerate graphs. It is $\mathcal{NP}$-hard to compute a feedback vertex set of minimum size [42] but since the size of a feedback vertex set is smaller than n and there are heuristics to compute a feedback vertex set in linear time, this parametrisation may still be of practical relevance.

Chordal A *chordal graph* is a graph in which each induced cycle is a triangle. Equivalently, a graph is chordal if and only if it has a *perfect elimination ordering*, that is, an ordering of the vertices such that for every vertex v it holds that all neighbours of v which come later in the perfect elimination ordering form a clique. Rose et al. [54] showed that a perfect elimination ordering, if one exists, can be computed in linear time using breadth-first search. Thus, chordal graphs may be recognised in linear time, too. Note that every subgraph of a chordal graph is chordal, too. Hence, being chordal is a hereditary property and thus we can use a result by Lewis and Yannakakis [47] which states that the minimum deletion set to Π is $\mathcal{NP}$-hard to compute when Π is defined by a hereditary property. It is, however, fixed-parameter tractable with respect to the distance to chordal graphs [48] and thus if the distance to chordal is small, then it is possible that using this parametrisation to compute both the minimum deletion set to chordal graphs and the number of triangles yield an algorithm that is faster than the existing unparametrised algorithms.

Bipartite A *bipartite graph* is a graph whose vertices can be partitioned into two independent sets. That is, there are two sets U, W of vertices with $U \cap W = \emptyset$ and $U \cup W = V$ such that each edge has one endpoint in U and one endpoint in W. Bipartite graphs can be recognised in linear time using depth-first search. Computing a minimum set K such that $G' = G - K$ is bipartite is $\mathcal{NP}$-hard since being bipartite is a hereditary property [47]. This parametrisation seems unpromising unless one has some prior knowledge that the distance to bipartite might be small. In this case, there are heuristics which compute a relatively small deletion set K' to bipartite graphs. We will show how to use this K' to find triangles in $O(n + |K'| \cdot m)$ time.

Cograph A *cograph* is a graph which can be constructed using the following three rules.

- A graph consisting of only a single vertex.
- The disjoint union of two cographs $F = (V_1, E_1)$, $H = (V_2, E_2)$. That is, $G = (V_1 \cup V_2, E_1 \cup E_2)$ where $V_1 \cap V_2 = \emptyset$.
- The join of two cographs $F = (V_1, E_1), H = (V_2, E_2)$. The join is defined as $G = (V_1 \cup V_2, E_1 \cup E_2 \cup \{\{u, v\} \mid u \in V_1 \land v \in V_2\})$ where $V_1 \cap V_2 = \emptyset$.

Every cograph G has a binary cotree representation. A *cotree* is a rooted forest T whose leaves represent the vertices in G and whose inner nodes each either represent a disjoint union or a join of the graphs induced by the leaves of their children. Corneil et al. [13] showed that cographs can be recognised in linear time by building a cotree representation, if any exists.

In order to achieve a full binary cotree for G, one can simply replace each inner node p which has more than two children by an arbitrary tree whose leaves are the children of p, each inner node has exactly two vertices and all represent the same operation as p. Hence, each node in T is either a leaf or has exactly two children and thus, T is a full binary cotree for G. Unfortunately, every subgraph of a cograph is a cograph. Hence being a cograph is a hereditary property and thus computing the distance to cographs is $\mathcal{NP}$-hard [47]. We currently do not have any information on how large the distance to cographs of real-world applications is and how fast it can be approximated.

Perfect A *perfect graph* is a graph G such that in any subgraph of G the size of a maximum-cardinality clique equals the chromatic number of that subgraph. Perfect graphs can be recognised in polynomial time and the distance to perfect graphs is $\mathcal{NP}$-hard to compute as being a perfect graph is a hereditary property. Since it takes more time to test whether a graph is perfect, than it takes to compute all triangles in this graph, it seems not promising to use this parametrisation.

Planar A finite graph is *planar* if and only if it can be drawn on the plane without a crossing of two edges. Planar graphs are 5-degenerate [4] and can be recognised in linear time [39].

Outerplanar A finite graph is *outerplanar* if and only if it can be drawn on the plane without a crossing of two edges such that all vertices belong to the outer face of the drawing. Outerplanar graphs are 2-degenerate [24] and can be recognised in linear time [50].

d-degenerate A *d-degenerate graph* is a graph whose degeneracy is d. For any fixed d the distance to d-degenerate graphs is $\mathcal{NP}$-hard to compute since every subgraph of a d-degenerate graph is a d-degenerate graph and thus being d-degenerate is a hereditary property. Fortunately, the distance to d-degenerate graphs is relatively small in many applications such as social networks because the degree distributions in these graphs follow a power law [25] and thus there are few vertices with high degree. Finding a d such that there is a good balance between the degeneracy d and the distance k to d-degenerate graphs is, from our point of view, one of the most efficient ways to find triangles as fast as possible.

2.4.2 Parameter Hierarchy

See Figure 4 for an overview on the parameter hierarchy [57]. Each box represents one parametrisation and each edge between two boxes represents a relation such that the lower parametrisation A is upper-bounded by the above parametrisation B. That is, there is a computable function f such that for any graph G the integer value $||a||$ associated with the parameter a of A in G is at most $f(||b||)$, where $||b||$ is the integer value associated with the parameter b of B in G.

Those relations can be used to obtain new results in two different ways. Let P be some fixed graph problem. First, assume there is an algorithm solving P parametrised by A for each input graph G in $g(||a||) \cdot h(|G|)$ time for some monotonically increasing function g and some computable function h. This is upper-bounded by $g(f(||b||)) \cdot h(|G|)$ and thus P parametrised by B can be solved in $g(f(||b||)) \cdot h(|G|)$ time. Second, assume there is no algorithm solving P parametrised by B for each input graph G in $g(||b||) \cdot h(|G|)$ time for a fixed function h and any computable function g. Assume towards a contradiction that there is an algorithm solving P parametrised by A for each input graph G in $g'(||a||) \cdot h(|G|)$ time for some computable function g'. Since $||a||$ is upper-bounded by $f(||b||)$, P parametrised by B would be solvable in $g'(f(||b||)) \cdot h(|G|)$ time, which is a contradiction. Therefore, there is no algorithm solving P parametrised by A in $g'(||a||) \cdot h(|G|)$ time. Summarizing, tractability transfers to parameters above

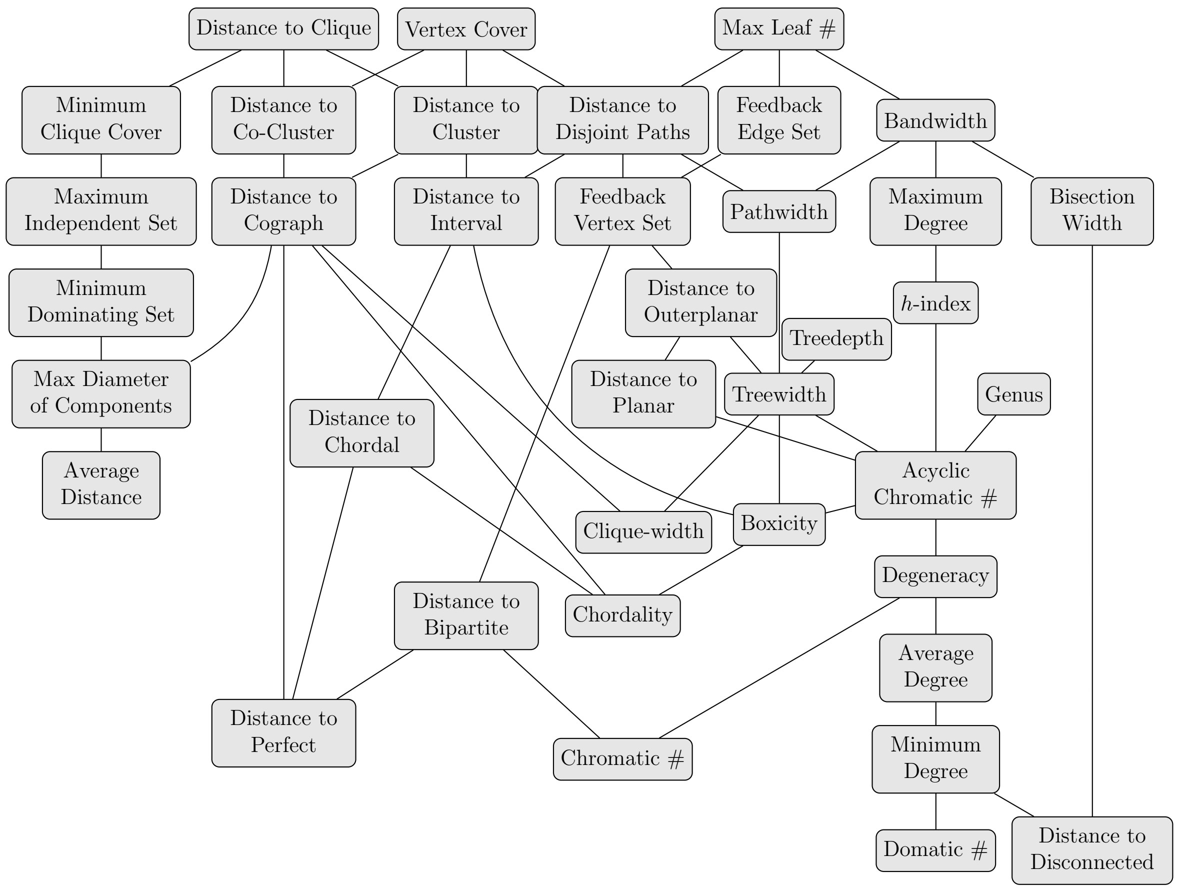

Figure 4: Parameter hierarchy [57]

and intractability to parameters below. Note that in this thesis we sometimes assume a specific parameter to be given. In this case a tractability result may not transfer to a higher parameter.

3 Algorithmic Results

In this section we provide algorithms to solve Triangle Detection, Triangle Counting, Triangle Enumeration, c-Clique and Girth with different parametrisations and we analyse their running times. In the following we always denote the value of the corresponding parameter by k and, if existing, the set associated with the parameter by K. In general, we do not assume K to be the optimum over all possible parameter sets and not even to be minimal (or maximal). For the sake of simplicity we omitted the special case of $k = 0$. This can easily be fixed by writing $k + 1$ instead of k, but for most parameters the special case can be solved in either linear time or even in constant time.

Recall that Proposition 2.2 and Observation 2.1 state that every algorithm which solves Triangle Enumeration can be used to solve Triangle Counting and each algorithm which solves Triangle Counting can be used to solve Triangle Detection. We therefore omit algorithms for Triangle Detection in the following if we can solve Triangle Enumeration as fast as Triangle Detection.

The outline of this section is as follows. We will present algorithms that use one parametrisation and have fast running times first and the worst-case running time of the presented algorithms will increase over the course of this section. All algorithms which use a certain parametrisation will be contained in the same subsection. In the last subsection, we will present an algorithm solving Triangle Enumeration parametrised by maximum degree Δ and deletion set to d-degenerate graphs of size k in $O(n \cdot d^2 + k \cdot \Delta^2)$ time. Green and Bader [34] showed that Triangle Counting can be solved in $O(k \cdot \Delta^2)$ where k is the size of a vertex cover. Note, that a vertex cover is a deletion set to 0-degenerate graphs. Note further, that our algorithm solves Triangle Enumeration rather than Triangle Counting. Our result can therefore be seen as a generalisation of the result by Green and Bader.

Figures 5 and 6 illustrate a condensed form of our results. Figure 5 shows our results on Triangle Detection and Figure 6 shows our results on Triangle Counting and Triangle Enumeration. We present algorithms and analyse their running times for all parametrisations that are coloured gray, lightgreen, yellow or orange. The other results can be deduced from these as shown in Section 2.4.2. Note that all results on Triangle Enumeration can also be used for Triangle Counting.

We want to especially highlight Theorems 3.8 and 3.25. These two results use different aspects of the parametrisation degeneracy to achieve fast running times. Batagelj and Zaversnik [7] showed how to compute the degeneracy in $O(n + m)$ time. Eppstein et al. [23] observed that the degeneracy of real-world graphs is often very small and therefore using the degeneracy parameter is well-motivated from a practical point of view.

3.1 Algorithms Independent of the Input Size

In this section we discuss parameters which can be used to construct algorithms solving Triangle Detection with a running time independent of the number of vertices and

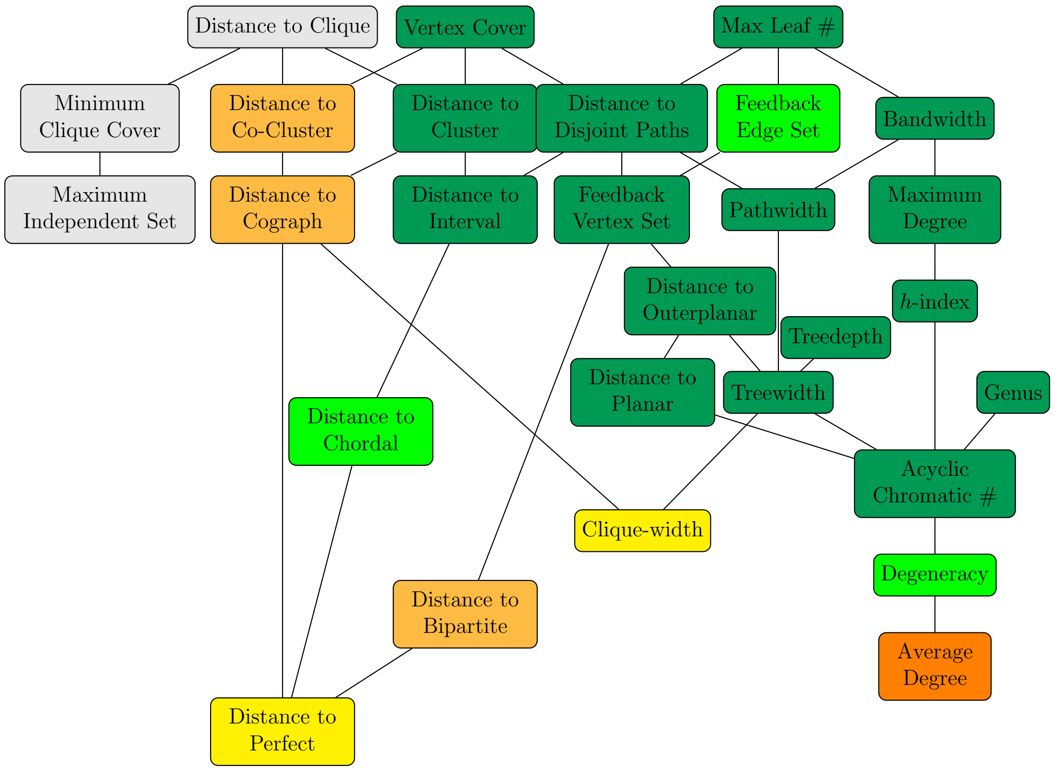

Figure 5: Parameter hierarchy and results for TRIANGLE DETECTION. Colouring is based on the running time the respective algorithms achieve. For more information see Sections 1.2 and 2.4.2. Recall, that n is the number of vertices and m is the number of edges in the input graph.
independent of n
linear in n
linear in $n + m$
quadratic in n
linear in n deduced from an other parameter
linear in $n + m$ deduced from an other parameter

edges. These parameters are distance to clique and independent set.

Recall that the distance to clique is the size of a set K of vertices one has to delete in G such that the resulting graph $G - K$ is a clique, that is, every two vertices are pairwise adjacent. We do not need the set K to be given, but only the size $k = |K|$. The following results on TRIANGLE DETECTION and c-CLIQUE assume the minimum distance to clique to be given.

Observation 3.1. TRIANGLE DETECTION *parametrised by minimum distance to clique is solvable in* $O(1)$ *time.*

Proof. We show that G contains a triangle if and only if the minimum distance to clique is smaller than $n - 2$. If there is a triangle in G, then the distance to this clique

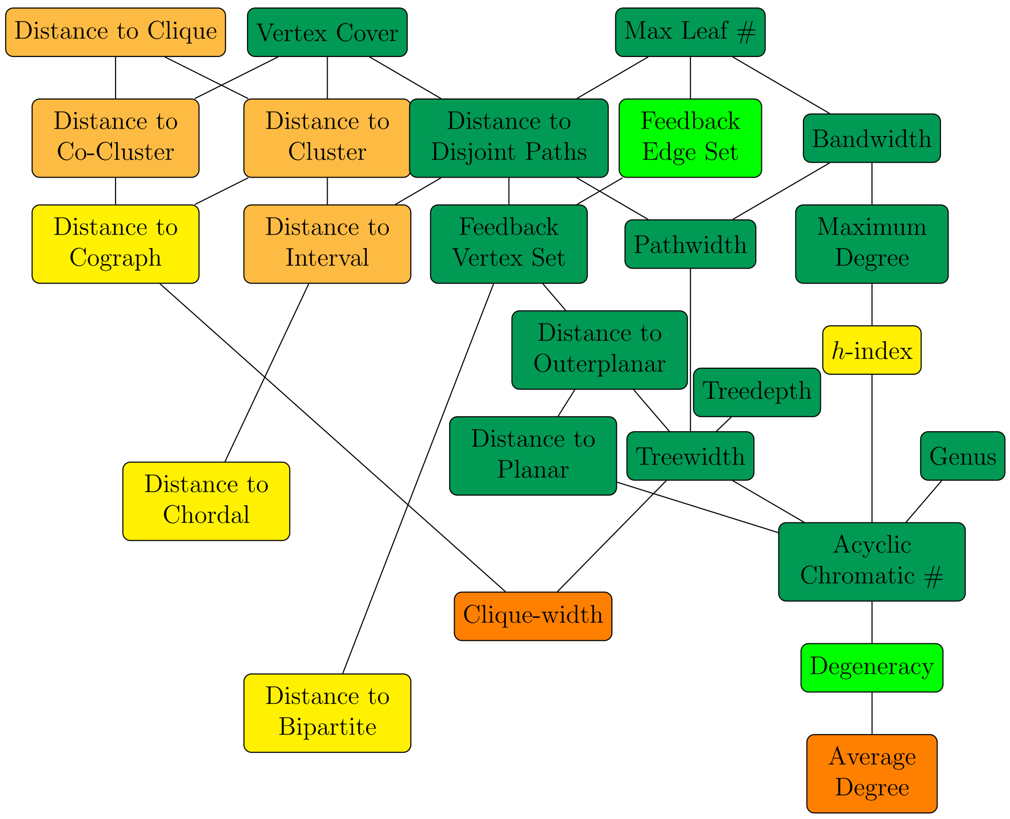

Figure 6: Parameter hierarchy and results for Triangle Counting and Triangle Enumeration. Colouring is based on the running time the respective algorithms achieve. For more information see Sections 1.2 and 2.4.2. Recall, that n is the number of vertices and m is the number of edges in the input graph.
linear in n
linear in $n + m$
quadratic in n
linear in n deduced from an other parameter
linear in $n + m$ deduced from an other parameter

is at most $n - 3$. If G is triangle-free, then every clique has size at most two and thus the distance to any clique is at least $n - 2$. This can be decided in constant time. □

Note that this result only requires knowledge about n and k and not the whole graph G. Thus, if the graph is given in a representation as shown in Figure 2 (which we assume throughout this thesis), then n and k can be read in constant time without reading the whole input and the algorithm takes $O(1)$ time. Otherwise one has to spend $O(n + m)$ time to read the input and compute the number n.

A straight-forward generalization of the argument behind Observation 3.1 yields the following.

Corollary 3.2. *c-*CLIQUE *parametrised by minimum distance to clique is solvable in $O(1)$ time.*

The girth of a graph is three if there is a triangle in G. We will use this to give a fast algorithm solving GIRTH parametrised by distance to clique. Note that we do not need the minimum distance this time.

Corollary 3.3. GIRTH *parametrised by distance k to clique is solvable in $O(k^3)$ time.*

Proof. If the distance k to clique is smaller than $n - 2$, then there is a triangle in G and therefore the girth is three. This can be decided in constant time. Otherwise it holds that $n \leq k + 2$ and $m \leq (k + 2)^2$. Thus, the algorithm presented by Itai and Rodeh [40] that takes $O(n \cdot (n + m))$ time to solve GIRTH runs in $O(k^3)$ time. □

We will now focus on algorithms using the maximum independent set size. Recall that an independent set is a set K of vertices such that all vertices in K are pairwise non-adjacent. The maximum independent set size is the largest number k such that there is an independent set of size k in the graph. We do not need a set K to be given, but only the size $k = |K|$ of a largest independent set. However, the following results require the maximum independent set size and not just the size of any maximal independent set.

We will first introduce a short lemma which we will use to construct an algorithm for TRIANGLE DETECTION parametrised by maximum independent set size. The lemma states that every triangle-free graph has an independent set of size at least $\sqrt{n}$. This can be seen via a case distinction over the maximum degree in G and the fact that the neighbourhood of any vertex is an independent set in triangle-free graphs.

Lemma 3.4 (folklore)**.** *If the maximum independent set size of a graph is smaller than $\sqrt{n}$, then the graph contains a triangle.*

Proof. We show that every triangle-free graph G has an independent set of size at least $\sqrt{n}$. If there is a vertex v with at least $\sqrt{n}$ neighbours, then these neighbours are either an independent set or there is an edge between two neighbours u and w and thus $\{u, v, w\}$ is a triangle. Since we assume G to be triangle-free, we have an independent set of size $\sqrt{n}$.

Now consider the case that there is no vertex of degree at least $\sqrt{n}$. In this case one can take an arbitrary vertex v, add it to the independent set, delete v and all of it neighbours from G and repeat until no vertex is left. As we delete at most $\sqrt{n}$ vertices in each iteration and add one vertex to the independent set, this independent set is of size at least $n/\sqrt{n} = \sqrt{n}$ after the last iteration.

Therefore, if the maximum independent set size is less than $\sqrt{n}$, then G contains a triangle. □

Recall that ω is the fast matrix multiplication exponent. We will make a case distinction over the maximum independent set size in order to solve TRIANGLE DETECTION parametrised by maximum independent set size in $O(k^{2\omega})$ time.

Corollary 3.5. Triangle Detection *parametrised by maximum independent set size k is solvable in $O(k^{2\omega})$ time.*

Proof. Lemma 3.4 shows that if the maximum independent set size k is smaller than $\sqrt{n}$, then the graph contains a triangle. Otherwise, it holds that $n \leq k^2$ and thus one can solve Triangle Detection in $O(n^{\omega}) \subseteq O(k^{2\omega})$ by an algorithm presented by Latapy [45] that solves Triangle Detection and Triangle Counting. □

3.2 Parametrisation by feedback edge number

Recall that a feedback edge set is a set K of edges such that removing all edges in K from G results in a forest, that is, $G - K$ is cycle-free. Furthermore recall that a minimum feedback edge set on an undirected unweighted graph can be computed in $O(n+m)$ time using depth-first search. Hence, the following results can be achieved even when K is not given. We will give two algorithms solving Triangle Enumeration parametrised by feedback edge number. Note that it might be that $|K| > |V|$ as K is a set of edges and not a set of vertices. Therefore the following two algorithms are incomparable in terms of their worst-case running times.

Theorem 3.6. Triangle Enumeration *parametrised by feedback edge number k is solvable in $O(k^2 + n + m)$ and in $O(k \cdot n + m)$ time.*

Proof. Let K be a feedback edge set in G with $|K| = k$ which can be computed in $O(n+m)$ time using depth-first search. Let $G' = (V, E \setminus K)$ be an arbitrarily rooted forest and let $p(v)$ denote the parent of v in G'. The graph G' can be computed in $O(n+m)$ time using breadth-first search and clearly G' is triangle-free. If G contains a triangle $\{v_x, v_y, v_z\}$, then at least one edge between any two of the three vertices is in K. Let, without loss of generality, be $\{v_x, v_y\} \in K$. We will present two algorithms which list all triangles in G. We will show that the triangle $\{v_x, v_y, v_z\}$ is found and since we do not make any assumptions about this specific triangle, any other triangle is listed, too. Algorithm 2 runs in $O(k^2+n+m)$ time and Algorithm 3 runs in $O(k\cdot n+m)$ time.

Algorithm 2 works as follows. If $\{v_x, v_z\}, \{v_y, v_z\} \notin K$, then $v_z = p(v_x)$ or $v_z = p(v_y)$. This is because v_z can be the child vertex of at most one of the two vertices v_x and v_y and thus at least the other one must be a child of v_z. Hence $\{v_x, v_y, v_z\}$ can be found in constant time by checking whether $\{v_x, p(v_y)\} \in E \setminus K$ or $\{p(v_x), v_y\} \in E \setminus K$. If this condition holds, then we can add this triangle to a set T of all triangles in G and list it. We assume this set T to be stored as a hash table in order to achieve constant amortized time per insert and lookup. Note that $p(v_x)$ might be the same as $p(v_y)$ in which case we only list the triangle once. Since $\{v_x, v_y\}$ is the only edge in this triangle which is also in K, we do not need to check if this triangle is already in T.

If $\{v_x, v_z\} \in K$ or $\{v_y, v_z\} \in K$, then $\{v_x, v_y, v_z\}$ can be found in time $O(k^2)$ by checking for any $e_1, e_2 \in K$ whether the union of all endpoints is a triangle. Formally, this is $e_1 \cap e_2 \neq \emptyset$ and $((e_1 \cup e_2) \setminus (e_1 \cap e_2)) \in E$. If any triangle is found, then we check whether this triangle is already in T and if not, then we insert it to T and list it.

Algorithm 2 Triangle Enumeration parametrised by feedback edge number k in $O(k \cdot n + m)$ time.

```
1: compute minimum feedback edge set $K$;
2: compute $G' = (V, E \setminus K)$;
3: compute parent function $p$;
4: $T := \emptyset$;
5: for $\{v_x, v_y\} \in K$ do
6:     if $p(v_x) = p(v_y)$ then list $\{v_x, v_y, p(v_x)\}$;    ▷ Cases: Exactly on edge in $K$
7:     else
8:         if $\{p(p(v_x)) = v_y\}$ then list $\{v_x, v_y, p(v_x)\}$;
9:         if $\{v_x = p(p(v_y))\}$ then list $\{v_x, v_y, p(v_y)\}$;
10:    for $\{v_z, v_w\} \in K$ do    ▷ Case: At least two edges in $K$
11:        if $v_x = v_z \wedge \{v_y, v_w\} \in E \wedge \{v_x, v_y, v_z\} \notin T$ then    ▷ Case: $v_x = v_z$
12:            list $\{v_x, v_y, v_w\}$;
13:            $T := T \cup \{\{v_x, v_y, v_w\}\}$;
14:        if $v_x = v_w \wedge \{v_y, v_z\} \in E \wedge \{v_x, v_y, v_z\} \notin T$ then    ▷ Case: $v_x = v_w$
15:            list $\{v_x, v_y, v_z\}$;
16:            $T := T \cup \{\{v_x, v_y, v_z\}\}$;
17:        if $v_y = v_z \wedge \{v_x, v_w\} \in E \wedge \{v_x, v_y, v_z\} \notin T$ then    ▷ Case: $v_y = v_z$
18:            list $\{v_x, v_y, v_w\}$;
19:            $T := T \cup \{\{v_x, v_y, v_w\}\}$;
20:        if $v_y = v_w \wedge \{v_x, v_z\} \in E \wedge \{v_x, v_y, v_z\} \notin T$ then    ▷ Case: $v_y = v_w$
21:            list $\{v_x, v_y, v_z\}$;
22:            $T := T \cup \{\{v_x, v_y, v_z\}\}$;
```

Algorithm 3 runs as follows. Again we assume $\{v_x, v_y\} \in K$ and we want to list the arbitrary triangle $\{v_x, v_y, v_z\}$.

For each $v_z \in V$ check whether $\{v_x, v_z\} \in E$ and $\{v_y, v_z\} \in E$ in constant time. If so, then check whether this triangle was already listed in constant amortized time using a hash table for T. If this triangle is not in T, then add it to T and list the triangle. As there are n vertices in V, k edges in K and one need $n + m$ time to read the input, Algorithm 3 runs in $O(n \cdot k + m)$ time. □

3.3 Parametrisation by Degree-Related Parameters

In this subsection we will discuss Triangle Counting parametrised by h-index and Triangle Enumeration parametrised by degeneracy or average degree.

3.3.1 Parametrisation by h-Index

Recall that the h-index of a graph is the largest number k such that there are at least k vertices with at least k neighbours. We use a dynamic data structure presented by

Algorithm 3 TRIANGLE ENUMERATION parametrised by feedback edge number k in $O(k^2 + n + m)$ time.

```
1: compute minimum feedback edge set $K$;
2: $T := \emptyset$;
3: for $\{v_x, v_y\} \in K$ do
4:     for $v_z \in V$ do
5:         if $\{v_x, v_z\} \in E \wedge \{v_y, v_z\} \in E \wedge \{v_x, v_y, v_z\} \notin T$ then
6:             list $\{v_x, v_y, v_z\}$;
7:             $T := T \cup \{\{v_x, v_y, v_z\}\}$;
```

Eppstein and Spiro [22] which can be used to solve TRIANGLE COUNTING parametrised by h-index in $O(m \cdot k + n)$ amortized time.

Proposition 3.7. TRIANGLE COUNTING *parametrised by h-index k is solvable in $O(m \cdot k + n)$ amortized time.*

Proof. We will use a dynamic data structure described by Eppstein and Spiro [22] to solve TRIANGLE COUNTING parametrised by h-index by building G from scratch and keep track of the number of triangles. Eppstein and Spiro described a dynamic data structure which maintains the number of triangles in G and requires $O(h)$ amortized time per operation, where h is the h-index at the time of the operation. An operation is either inserting a new vertex, inserting a new edge or deleting an existing edge.

We will now show how the graph G can be converted in this data structure in $O(m \cdot h + n)$ time. First add n vertices consecutively to the empty graph represented in the described data structure and then add all edges of G to this newly constructed graph. Note that the h-index of G is still one after inserting all vertices. Hence, all vertices can be inserted in $O(n)$ time and all edges can be inserted in $O(m \cdot h)$ time. Once finished, one can simply read the number of triangles in G from the data structure and compare it to the threshold t in constant time. □

3.3.2 Parametrisation by Degeneracy

Recall that the degeneracy of a graph is the smallest number k for which there is an order $\leq_d$ such that for every vertex v the number of neighbours w of v with $v \leq_d w$ is at most k. Furthermore, recall that both this order and an ordering of the vertices $v_1, \dots, v_n$ with $v_i \leq_d v_{i+1}$ for all $i \in [1, n-1]$ can be computed in $O(n + m)$ time. We will use this to enumerate all triangles $\{u, v, w\}$ for a vertex v in which $v \leq_d u$ and $v \leq_d w$.

Theorem 3.8. TRIANGLE ENUMERATION *parametrised by degeneracy k is solvable in $O(n \cdot k^2)$ time.*

Proof. We will present an algorithm solving TRIANGLE ENUMERATION parametrised by degeneracy k in $O(n \cdot k^2)$ time. Algorithm 4 works as follows. First order the vertices and relabel them such that $v_i \leq_d v_{i+1}$ holds for all $i \in [1, n-1]$. This can be done

Algorithm 4 TRIANGLE ENUMERATION parametrised by degeneracy k in $O(n \cdot k^2)$ time.

```
1: compute the degeneracy ordering $v_1, \dots, v_n$;
2: for $i := 1$ to $n$ do
3:     for $v_j \in N(v_i)$ do
4:         for $v_k \in N(v_i)$ do
5:             if $\{v_j, v_k\} \in E \wedge j < k$ then list $\{v_i, v_j, v_k\}$
6:     Delete $v_i$ and all edges which have $v_i$ as an endpoint;
```

in $O(n+m)$ as shown by Batagelj and Zaversnik [7]. Repeat the following steps until no vertex is left in G:

Take the vertex v which is currently the first in the ordering and enumerate all triangles which contain v in $O(k^2)$ time. This can be done by checking for each $u, w \in N(v)$ whether $\{u, w\} \in E$. If so, then $\{u, v, w\}$ is a triangle as both u and w are adjacent to v. Since v is the first vertex in the degeneracy order it holds that $|N(v)| \leq k$. Hence there are at most k^2 possible choices to pick two vertices from $N(v)$. After checking all possible combinations of neighbours of v, delete v as this vertex cannot be part of any further triangles. The ordering of vertices which come later in the order is not affected by v.

Since every vertex is at some point the first in the degeneracy order, we can conclude that we checked for all possible triangles and thus all triangles are listed.

This algorithm runs in $O(n \cdot k^2)$ time, because each iteration takes $O(k^2)$ time to check for triangles and there are n iterations. Deleting all vertices and all edges can be done in $O(n+m)$ time and since average degree is upper-bounded by degeneracy, it holds that $2m \leq n \cdot k$. $\square$

Note that the degeneracy order can be computed without any prior knowledge about the degeneracy of the graph. Therefore this algorithm solves TRIANGLE ENUMERATION parametrised by any parameter k which upper-bounds degeneracy in $O(n \cdot k^2)$ time.

A straight-forward generalisation of the idea behind Theorem 3.8 leads to the following.

Corollary 3.9. *c-CLIQUE parametrised by degeneracy k is solvable in $O(n \cdot k^{c-1} \cdot c^2)$ time.*

Proof. Compute the degeneracy order $\leq_d$ and an ordering $v_1, \dots, v_n$ of the vertices in $O(n+m)$ time such that $v_i \leq v_{i+1}$ holds for all $i \in [1, n-1]$. Take the first vertex v in the ordering, check all possible combinations of $c-1$ of its neighbours for a clique of size c and delete v afterwards. Repeat until no vertex is left or until a clique of size c is found. If a clique of size c is found, then return yes and if no vertex is left, then return no. As v has at most k neighbours at each point, we can compute all cliques of size c which contain v in $O(k^{c-1} \cdot c^2)$ time by trying all possible combinations of $c-1$ of its neighbours and checking in $O(c^2)$ time whether this is a clique. As there are at most n iterations, this algorithm takes $O(n \cdot k^{c-1} \cdot c^2)$ time. $\square$

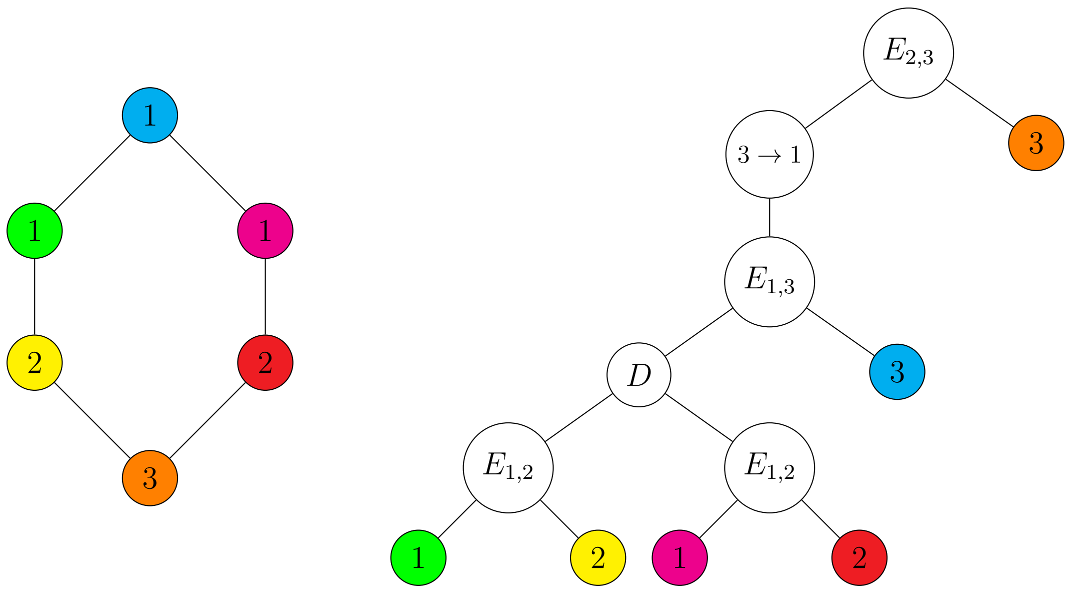

Figure 7: Example of a graph and its k-expression where $k = 3$.

3.3.3 Parametrisation by Average Degree

Itai and Rodeh [40] showed that Triangle Counting can be solved in $O(m^{1.5})$ time. Note that we used their algorithm already in the proof of Reduction Rule 2.4. It is straight-forward to modify their algorithm in order to enumerate triangles instead of counting them. As the average degree k is equal to $m/2n$, $O(m^{1.5})$ is the same as $O(n^{1.5} \cdot k^{1.5})$, and thus we arrive at the following.

Proposition 3.10. Triangle Enumeration *parametrised by average degree k is solvable in $O(n^{1.5} \cdot k^{1.5})$ time.*

3.4 Parametrisation by Clique-Width

Recall that the clique-width of a graph G is the minimum number k such that G can be constructed using a k-expression. A k-expression consists of four operations which use k labels [14]. The operations are the following.

- Creating a new vertex with some label i. (i)
- Disjoint union of two labeled graphs. (D)
- Edge insertion between every vertex with label i to every vertex with label j for some labels $i \neq j$. ($E_{i,j}$)
- Renaming of label i to j for some i, j. ($i \to j$)

Figure 7 shows an example of a graph and one of its k-expressions.

Gurski [36] has proven that the size of the largest clique in a graph can be computed in linear time when the graph is given by a k-expression. He assumes k to be a constant and therefore this k does not appear in the analysis of the running time. We will use this result to solve c-Clique in $f(k) \cdot (n+m)$ time where f is some computable function.

Proposition 3.11 (Gurski [36])**.** *Given a k-expression of the graph G, c-*Clique *is solvable in $f(k) \cdot (n+m)$ time for some computable function f.*

Due to the fact that a graph contains a triangle if and only if the largest clique in the graph is of size at least three, Triangle Detection can be solved in the same amount of time.

Corollary 3.12. *Given a k-expression of the graph G,* Triangle Detection *is solvable in $f(k) \cdot (n+m)$ time for some computable function f.*

Next, we will construct a dynamic program to solve Triangle Enumeration parametrised by some k-expression. To this end we use an ℓ-module decomposition. Let $V_1 \subseteq V$ be a set of vertices. A *twin-class* in V_1 is a set $V' \subseteq V_1$ of vertices such that every vertex in $V \setminus V_1$ either has all vertices in V' as its neighbours or none of them. The set V_1 is an *ℓ-module* if it can be partitioned into at most ℓ twin-classes. Let T be a rooted full binary tree whose leaves are in bijection to the vertices in V. For each inner node p in T let $V_p \subseteq V$ be the set of all vertices in G whose corresponding nodes in T are in the induced subtree of T rooted at p. If for each inner node p in T the set V_p is an ℓ-module, then T is an *ℓ-module decomposition.* Given some k-expression of G, a k-module decomposition of G and the twin-classes of all inner nodes can be computed in $O(n^2)$ time as shown by Bui-Xuan et al. [10]. For more information on k-modules see Vatshelle [59]. We will use this k-module decomposition to solve Triangle Enumeration parametrised by some k-expression in $O(k^2 \cdot n^2 + \#\mathrm{T})$ time.

Theorem 3.13. *Given a k-expression of the graph G,* Triangle Enumeration *is solvable in $O(k^2 \cdot n^2 + \#\mathrm{T})$ time where $\#\mathrm{T}$ is the number of triangles in G.*

Proof. We give an algorithm consisting of three phases. In the first phase, it computes a k-module decomposition of G. Then, in the second phase, it modifies the twin-classes in a way that certain conditions hold. Finally, in the third phase it computes all triangles using the decomposition. See Algorithm 5 for pseudo-code.

In the first phase we use a result by Bui-Xuan et al. [10, Lemma 3.2] which states that given a k-expression tree T of G, one can compute in overall $O(n^2)$ time for every node u in T the partition of V_u into its twin-classes $Q_u(1), Q_u(2), \cdots, Q_u(h_u)$. Here, V_u is the set of vertices corresponding to the leaves of the subtree of T rooted at u. Bui-Xuan et al. also proved that the maximum number of twin-classes h_u for each node u in T is at most k. They also showed how to modify T such that it becomes a full binary tree and thus combined with the twin-classes becomes a k-module decomposition of G. This completes the description of the first phase.

In the second phase we want to make sure that every twin-class $Q_u(a)$ is fully contained in one of the twin-classes $Q_v(b)$ when v is the parent of u in T and that the root

Algorithm 5 TRIANGLE ENUMERATION parametrised by a k-expression in $O(k^2 \cdot n^2 + \#\mathrm{T})$ time.

```
1: compute the expression tree $T$ of the $k$-expression;                ▷ phase one
2: contract all nodes in $T$ which have exactly one child;
3: let $p(x)$ denote the parent of $x$ in $T$;
4: compute for every node $u$ in $T$ the twin-classes $Q_u(1), \cdots, Q_u(h_u)$;
5: for $u$ in $T$ do                                                    ▷ phase two
6:     if $u$ is the root node in $T$ then              ▷ Iterate from leaves up to the root
7:         $Q_u(1) := V$;
8:         for $i := 2$ to $h_u$ do
9:             $Q_u(i) := \emptyset$;
10:    else
11:        for $a := 1$ to $h_u$ do
12:            $b := 0$;
13:            found $:= false$;
14:            while $\neg$found do
15:                $b := b + 1$
16:                if $Q_u(a) \cap Q_{p(u)}(b) \neq \emptyset$ then
17:                    $Q_{p(u)}(b) := Q_{p(u)}(b) \cup Q_u(a)$;
18:                    for $c := b + 1$ to $h_{p(u)}$ do
19:                        $Q_{p(u)}(c) := Q_{p(u)}(c) \setminus Q_u(a)$;
20:                    $M_{u,b} := M_{u,b} \cup \{a\}$;
21:                    found $:= true$;
22: compute all triangles by the following dynamic program;             ▷ phase three
23: for $v$ in $T$ do
24:     if $v$ is not a leaf in $T$ then
25:         look up children $u, w$ of $v$;                ▷ Needed for $E^v_{i,j}$ and $LT^v$
26:     for $i := 1$ to $h_v$ do
27:         for $j := 1$ to $h_v$ do
28:             compute $E^v_{i,j}$ and $LT^v$;             ▷ See Equations (1) to (4)
29: look up the root node $x$ in $T$
30: list all triangles in $LT^x$
```

node has only one twin-class. This can be done in $O(k^2 \cdot n^2)$ time by checking in $O(n \cdot k)$ time for each node u in T and each of its twin-classes $Q_u(a)$ which twin-class $Q_v(b)$ of the parent v of u contains an arbitrary vertex in $Q_u(a)$. If $Q_v(b)$ is the twin-class, then insert all vertices of $Q_u(a)$ into $Q_v(b)$ and delete all of those vertices from all other twin-classes of v in $O(n \cdot k)$ time using an ordered merge algorithm. Once this is done for one of the twin-classes, we can store the information that $Q_v(b)$ contains all vertices in $Q_u(a)$ for later use. Let $M_{u,b}$ be the set of all indices i such that all vertices in $Q_u(i)$ are contained in the twin-class $Q_v(b)$ of the parent v. The root node r can always have a single twin-class which contains all vertices in G as there are no vertices in $V \setminus V_r = \emptyset$

to distinguish between vertices in V_r.

In the third phase, we give a dynamic program to solve TRIANGLE ENUMERATION based on this newly constructed k-module decomposition. For each node v in T store $Q_v(1), \dots, Q_v(h_v)$, the twin-classes of v (which are already computed), $E^v_{i,j}$, all edges between vertices in twin-classes $Q_v(i)$ and $Q_v(j)$ $(1 \le i, j \le h_v)$, and LT^v, the set of all triangles formed by vertices in V_v.

Denote by $F_{u,a,w,b}$ the set of all edges with one endpoint in $Q_u(a)$ and one in $Q_w(b)$ where u and w have the same parent in T. This is

$$F_{u,a,w,b} = \begin{cases} \{\{x,y\} \mid x \in Q_u(a) \land y \in Q_w(b)\} & \text{if } \exists x \in Q_u(a), y \in Q_w(b).\{x,y\} \in E \\ \emptyset & \text{otherwise} \end{cases}$$

Note that by the definition of twin-classes, either all vertices of the twin-classes of two nodes, which have the same parent in T, are pairwise connected or none of them are.

The dynamic program is defined as follows. For a less formal definition of the equations to come see the first paragraph afterwards.

- A leaf v in T has for each $1 \le i, j \le h_v$:

$$E^v_{i,j} = \emptyset; \tag{1}$$
$$LT^v = \emptyset. \tag{2}$$

- An inner node v with children u, w has for each $1 \le i, j \le h_v$:

$$\begin{aligned} E^v_{i,j} = & \bigcup_{l \in M_{u,i}} \bigcup_{m \in M_{u,j}} E^u_{l,m} \cup \bigcup_{l \in M_{w,i}} \bigcup_{m \in M_{w,j}} E^w_{l,m} \cup \\ & \bigcup_{m \in M_{u,i}} \bigcup_{l \in M_{w,j}} F_{u,m,w,l} \cup \bigcup_{m \in M_{u,j}} \bigcup_{l \in M_{w,i}} F_{u,m,w,l}; \end{aligned} \tag{3}$$
$$LT^v = LT^u \cup LT^w \cup LT_{u,u,w} \cup LT_{w,w,u}, \tag{4}$$

where

$$\begin{aligned} LT_{x,x,y} = & \bigcup_{o=1}^{h_x} \bigcup_{p=o}^{h_x} \bigcup_{q=1}^{h_y} \{\{a,b,c\} \mid a \in Q_x(o) \land b \in Q_x(p) \land c \in Q_y(q) \land \\ & F_{x,o,y,q} \neq \emptyset \land F_{x,p,y,q} \neq \emptyset \land \{a,b\} \in E^x_{o,p}\}. \end{aligned}$$

The edges for an inner node are the edges of its child-nodes x, y and the union over all possible $F_{x,i,y,j}$. The triangles are those of its children plus all new ones $(LT_{x,x,y})$. Each of these new triangles $\{a,b,c\}$ have to have at least one vertex in x and at least one vertex in y. Let without loss of generality be $a \in Q_x(o), b \in Q_x(p)$ and $c \in Q_y(q)$. The set $\{a,b,c\}$ is a triangle if and only if $\{a,b\}, \{a,c\}, \{b,c\} \in E$. This is the same as $\{a,b\} \in E^x_{o,p}$, $F_{x,o,y,q} \neq \emptyset$ and $F_{x,p,y,q} \neq \emptyset$.

We will first analyse the running time for the presented algorithm and then we will prove that all triangles are listed.

There are n leaves in T and $n-1$ inner nodes. The construction above for leaves can be computed in constant time.

For each inner node, at most k^2 sets of edges have to be computed, each of which is built out of two parts: edges already stored in the edges of children in T and new edges between vertices of different children. The former requires $O(k^2)$ time per node as it can be seen as a list of pointers to the children's sets of edges. The latter requires $O(m)$ global time as each edge is only added once. Therefore the edge sets of all nodes can be computed in $O(n \cdot k^2 + m)$ time.

The list of triangles is a list containing two pointers to its children's list of triangles and a third list of new triangles. As each triangle is only added once, all lists can be computed in $O(\#\mathrm{T} + n)$ time.

We will prove that each triangle $\{x, y, z\}$ is found in their least ancestor node p. As each node in T references the triangles of its children, $\{x, y, z\}$ is passed on to the root node in T. Note that each node in T only computes those new triangles which have one vertex in the subtree rooted in one child node and two vertices in the subtree rooted in the other child node. Each triangle is therefore computed at most once.

Let q, r be the children of p and let without loss of generality be $x \in Q_q(s), y \in Q_q(t)$ and $z \in Q_r(u)$. Due to the fact that $\{x, z\}, \{y, z\} \in E$ it holds that $F_{q,s,r,u} \neq \emptyset$ and $F_{q,t,r,u} \neq \emptyset$. Moreover, it holds that $\{x, y\} \in E^q_{s,t}$ because $x \in Q_q(s), y \in Q_q(t)$ and $\{x, y\} \in E$. Thus, $\{x, y, z\}$ is contained in

$$\{\{a, b, c\} \mid a \in Q_q(s) \wedge b \in Q_q(t) \wedge c \in Q_r(u) \wedge F_{q,s,r,u} \neq \emptyset \wedge F_{q,t,r,u} \neq \emptyset \wedge \{a, b\} \in E^q_{s,t}\},$$

which is a subset of $LT_{q,q,r}$ which itself is a subset of LT^p. □

3.5 Parametrisation by Deletion Set to Π

In this subsection we will present several algorithms which use deletion set to Π parameters, that is, a set of vertices K such that $G - K$ is a graph in a pre-specified graph class Π. In particular, we will have a closer look at the classes of bipartite graphs, chordal graphs, cographs and perfect graphs. In the following, we assume the set K to be given as a parameter, but we do not need K to be the minimum set such that $G - K$ is in Π.

First we will present algorithms which solve Triangle Detection and afterwards we will show results on Triangle Enumeration. Algorithms which solve Girth or c-Clique will be presented after other algorithms which use a similar idea.

Recall that chordal graphs are those in which every induced cycle is a triangle. Thus, a chordal graph contains a triangle if and only if it contains a cycle of any length, that is, if it is not a forest. This observation yields the following.

Proposition 3.14. Triangle Detection *parametrised by deletion set K to chordal graphs is solvable in $O(n \cdot |K|^2)$ time.*

Proof. Let K be a set of vertices such that $G' = G - K$ is a chordal graph. A chordal graph contains a triangle if and only if it is not a forest. This can be checked using breadth-first search and takes $O(n)$ time as in each step either a new vertex or a feedback edge is found. If a feedback edge is found, then the algorithm terminates, since it just proved that the graph is not a forest. If no feedback edge is found, then in each step a new vertex is found and thus the algorithm takes at most n steps.

If G' is not a forest, then it contains a cycle and thus it contains a triangle. Otherwise, G' has degeneracy one. Hence, there is an order $\leq_d$ such that each vertex v has at most one neighbour w with $v \leq_d w$. By inserting all vertices in K to the end of this order, one gets a new order $\leq_{d'}$ such that each vertex v has at most $1 + |K|$ neighbours w with $v \leq_{d'} w$ in G and hence G has degeneracy $|K| + 1$. By Theorem 3.8, we can detect a triangle in $O(n \cdot |K|^2)$ time, if there is any. □

Next we will investigate algorithms using a deletion set to perfect graphs as parametrisation. Recall that in perfect graphs the chromatic number and the size of the largest clique are equal. This observation leads to the following lemma which we will use to solve TRIANGLE DETECTION parametrised by deletion set to perfect graphs afterwards.

Lemma 3.15. *Every triangle-free perfect graph is bipartite.*

Proof. A triangle-free graph contains no clique larger than two. By definition triangle-free perfect graphs are two-colourable and thus bipartite. □

Proposition 3.16. TRIANGLE DETECTION *parametrised by deletion set K to perfect graphs is solvable in $O(n + m \cdot |K|)$ time.*

Proof. Let K be a set of vertices such that $G' = G - K$ is a perfect graph. Every triangle-free perfect graph is bipartite (Lemma 3.15). Checking whether a graph is bipartite can be done in $O(n + m)$ time using breadth-first search.

If G' is not bipartite, then it contains a triangle and the algorithm can terminate. Otherwise, G' does not contain any triangle and hence each triangle in G has to contain at least one vertex in K. All triangles with at least one vertex $v \in K$ can be found in $O(m \cdot k)$ time by checking for each $\{u, w\} \in E$ whether $\{u, v, w\}$ is a triangle. Thus, one can solve TRIANGLE DETECTION parametrised by deletion set to perfect graphs in $O(n + m \cdot |K|)$ time. □

Next we will use Lemma 3.15 to solve GIRTH parametrised by deletion set to perfect graphs. Recall that a perfect graph either contains a triangle or it only contains cycles of even length. GIRTH can be solved on perfect graphs in $O(n^2)$ time by first checking in $O(n + m)$ time whether the graph is bipartite or it contains a triangle. If it is bipartite, then compute the girth in $O(n^2)$ time as shown by Itai and Rodeh. This yields the following.

Proposition 3.17. GIRTH *parametrised by deletion set K to perfect graphs is solvable in $O(n^2 + |K| \cdot (n + m))$ time.*

Algorithm 6 GIRTH parametrised by deletion set K to perfect graphs in $O(n^2 + k \cdot (n+m))$ time.

```
1: let $K$ denote a deletion set to perfect graphs;
2: fix an arbitrary order $\leq_a$ in $K$;
3: if $G - K$ is bipartite then
4:     compute the length $g$ of the shortest cycle in $G - K$;
5: else
6:     return 3;
7: for $i := 1$ to $k$ do
8:     lookup $i^{th}$ vertex $v_i$ of $K$;
9:     compute length $g'$ of a shortest cycle in $G$ through $v_i$ using breadth-first search;
10:    if $g' < g$ then
11:        $g := g'$;
12: return $g$;
```

Proof. Let K be a set of vertices such that $G' = G - K$ is a perfect graph. Every triangle-free perfect graph is bipartite (Lemma 3.15). Checking whether a graph is bipartite can be done in $O(n+m)$ time using breadth-first search.

If G' is not bipartite, then the girth of G is three. Otherwise all cycles in G' have even length and therefore GIRTH can be solved on G' in $O(n^2)$ time as shown by Itai and Rodeh [40].

For each vertex $v \in K$ we can compute a lower bound on the length of a shortest cycle containing v in $O(n+m)$ time using breadth-first search. For more information and a description of the algorithm we refer to Itai and Rodeh [40].

The girth of G is the length of the shortest cycle in G. This is the length of the shortest cycle in G' or the length of the shortest cycle containing at least one vertex in K. Therefore comparing the length of the shortest cycle in G' to all the lengths of the shortest closed walks through a vertex in K and taking the minimum results in the girth of G. This results in an $O(n^2 + k \cdot (n+m))$-time algorithm to solve GIRTH parametrised by deletion set to perfect graphs. See Algorithm 6 for pseudo-code. □

Grötschel et al. [35] have shown that the largest clique in a perfect graph can be found in polynomial time. We will use this to give an algorithm that is slightly faster than $O(n^c)$ time for c-CLIQUE parametrised by deletions set to perfect graphs. The main idea is a case distinction whether the largest clique contains a vertex in K or not. If not, then the clique can be found in polynomial time. Otherwise a clique of size c can be found in $O(n^{c-1} \cdot k \cdot c^2)$ time, if there is any, by trying all possible combinations.

Proposition 3.18. *c-CLIQUE parametrised by deletion set K to perfect graphs is solvable in $O(n^{c-1} \cdot |K| \cdot c^2 + (n+m)^{O(1)})$ time.*

Proof. Let K be a set such that $G' = G - K$ is perfect. Grötschel et al. [35, Corollary 9.3.33] stated that c-CLIQUE can be solved in polynomial time on perfect graphs. If the size of the largest clique in G' is smaller then c, then all cliques in G of size at least c

have to include at least one vertex in K. All those cliques can be found in $O(n^{c-1} \cdot k \cdot c^2)$ time by simply checking for each vertex $v \in K$ and each possible combinations of $c-1$ vertices $S \subset V$ in $O(c^2)$ time whether $S \cup \{v\}$ is a clique. $\square$

We will now have a look at algorithms solving Triangle Enumeration. As many arguments have a similar structure, we will first prove a short lemma which will become useful later on. The idea is to solve Triangle Enumeration on graphs in Π in time x and enumerate all triangles with at least one vertex in K in $O(|K| \cdot m + n)$ time afterwards.

Lemma 3.19. Triangle Enumeration *parametrised by deletion set K to Π is solvable in $O(m \cdot |K| + n + x)$ time if* Triangle Enumeration *on a graph in Π is solvable in $O(x)$ time.*

Proof. Let K be a set of vertices such that $G' = G - K$ is a graph in Π. By assumption, all triangles within G' can be listed in $O(x)$ time. All triangles with at least one vertex in K can be listed in $O(m \cdot |K| + n)$ time by the following algorithm. Read the whole input and fix an arbitrary order $\leq_a$ of the vertices in K in $O(n + m)$ time. Check for each edge $\{u, w\}$ and each vertex $v \in K$ whether $\{u, v, w\}$ is a triangle and whether for all $x \in \{u, w\} \cap K$ it holds that $v \leq_a x$. If both conditions hold, then list $\{u, v, w\}$ as a new triangle. We will prove that this algorithm lists all triangles with at least one vertex in K exactly once. Since $v \in K$ holds, this algorithm does not list any triangles which do not contain vertices in K. Let $\{a, b, c\}$ be an arbitrary triangle and let a be in K. This triangle is found at least once as $\{b, c\} \in E$ and $a \in K$ holds. If for all $x \in \{b, c\}$ it holds that $x \notin K$ or $a \leq_a x$, then this triangle is listed in the iteration where $v = a$ and $\{u, w\} = \{b, c\}$. Otherwise, $b \leq_a a$ or $c \leq_a a$ holds. Let, without loss of generality, b be the first vertex out of $\{a, b, c\}$ in the fixed order. Then $\{a, b, c\}$ is listed in the iteration where $v = b$ and $\{u, w\} = \{a, c\}$ holds. There are $m \cdot |K|$ iterations and each iteration takes constant time. Therefore Triangle Enumeration parametrised by deletion set to Π is solvable in $O(m \cdot |K| + n)$ time. $\square$

Recall that bipartite graphs do not contain cycles of odd length and thus are triangle-free.

Corollary 3.20. Triangle Enumeration *parametrised by deletion set K to bipartite graphs is solvable in $O(n + m \cdot |K|)$ time.*

Proof. A bipartite graph does not contain any triangle. Hence, by Lemma 3.19, Triangle Enumeration parametrised by deletion set K to bipartite graphs is solvable in $O(n + m \cdot |K|)$ time. $\square$

As bipartite graphs do not contain triangles, they do not contain larger cliques either. Thus, any clique can contain at most two vertices out of any bipartite subgraph.

Corollary 3.21. c-Clique *parametrised by deletion set K to bipartite graphs is solvable in $O(n + m \cdot |K|^{c-2} \cdot c^2)$ time.*

Proof. Let K be a set of vertices such that $G' = G - K$ is a bipartite graph. Each clique can contain at most one vertex out of each partition in G' as each partition is an independent set. Hence, at most two vertices in each clique in G can be outside of K and there has to be an edge between those two. Thus, checking for each edge $\{u, v\} \in E$ and each subset $S \subset K$ with $|S| = c - 2$ if $S \cup \{u, v\}$ is a clique of size c solves c-Clique parametrised by deletion set to bipartite graphs in $O(n + m \cdot k^{c-2} \cdot c^2)$ time. Note that u, v can be in K. □

Recall that for each chordal graph there is a perfect elimination order $\leq_p$ of the vertices which can be computed in linear time. That is, for each vertex v all neighbours w of v with $v \leq_p w$ are pairwise adjacent. Moreover, recall that cographs have a binary cotree representation which can be computed in linear time as shown by Corneil et al. [13]. A cotree is a rooted tree in which each leaf corresponds to a vertex in the cograph and each inner node either represents a disjoint union or a join of its children. A join of two graphs $(V_1, E_1), (V_2, E_2)$ with $V_1 \cap V_2 = \emptyset$ is the graph $(V_1 \cup V_2, E_1 \cup E_2 \cup \{\{x, y\} \mid x \in V_1 \wedge y \in V_2\})$.

We use these representations to find all triangles in chordal graphs and cographs in $O(\#\mathrm{T} + n + m)$ time and thus by Lemma 3.19 solve Triangle Enumeration parametrised by deletions set K to chordal graphs or cographs in $O(\#\mathrm{T} + n + m \cdot |K|)$ time. Recall that $\#\mathrm{T}$ is the number of triangles in G. Note that a clique containing n vertices contains $O(n^3)$ triangles and that graphs consisting of only one large clique are both chordal graphs and cographs. Therefore we cannot avoid the term $\#\mathrm{T}$ in the running time.

Theorem 3.22. Triangle Enumeration *parametrised by deletion set K to chordal graphs or cographs is solvable in $O(\#\mathrm{T} + n + m \cdot |K|)$ time where $\#\mathrm{T}$ is the number of triangles in G.*

Proof. We will show that Triangle Enumeration is solvable in $O(\#\mathrm{T} + n + m)$ time on chordal graphs and on cographs. The statement of the theorem then follows from Lemma 3.19.

Chordal graphs: Compute the perfect elimination order in $O(n + m)$ time. Once this order is computed one can use it to list all triangles containing the first vertex in this order, delete it afterwards and continue until no vertex is left. Listing all triangles which contain the first vertex v in the perfect elimination order can be done as follows. As v is the first vertex in the order there are no vertices before v and hence $N(v)$ is a clique. Therefore v combined with any two of its neighbours forms a triangle. Formally, this is $\{\{v, x, y\} \mid x \in N(v) \wedge y \in N(v) \wedge x \neq y\}$. Once all of these triangles are listed, v is not contained in any more triangles and therefore one can delete it.

Cographs: Recall that every cograph has a binary cotree representation that can be computed in $O(n + m)$ time as shown by Corneil et al. [13]. Consider a dynamic program which stores for each node p in the cotree all vertices $V(p)$, all edges $E(p)$ and all triangles $T(p)$ in the corresponding subgraph of G. This can be done as follows:

Let q_1, q_2 be the children of an inner node p in the cotree.

- A single leaf node has one vertex and no edges or triangles.

- A union node has vertices $V(q_1) \cup V(q_2)$, edges $E(q_1) \cup E(q_2)$ and triangles $T(q_1) \cup T(q_2)$.

- A join node has

$$\begin{aligned}
V(p) &= V(q_1) \cup V(q_2),\\
E(p) &= E(q_1) \cup E(q_2) \cup \{\{x, y\} \mid x \in V(q_1) \wedge y \in V(q_2)\} \text{ and}\\
T(p) &= T(q_1) \cup T(q_2) \cup \{\{x, y, z\} \mid x \in V(q_1) \wedge \{y, z\} \in E(q_2)\} \cup\\
&\qquad \{\{x, y, z\} \mid x \in V(q_2) \wedge \{y, z\} \in E(q_1)\}.
\end{aligned}$$

That is, a join node contains all the edges the two children contain and all possible edges between vertices of them. A join node contains all triangles its two child-nodes contain and one triangle for each edge $\{y, z\}$ of one of its children and a vertex x of the other, because edges $\{x, y\}$ and $\{x, z\}$ are in E and therefore $\{x, y, z\}$ is a triangle.

We will first prove that all triangles are enumerated that way and afterwards we will analyse the running time of the dynamic program.

Let $\{a, b, c\}$ be any triangle in the cograph. We will prove that there is at least one node p in the cotree with $\{a, b, c\} \in T(p)$. As each inner node keeps the triangles from its children, it follows that $\{a, b, c\} \in T(r)$ when r is the root node of the cotree. Let, without loss of generality, p be the least common ancestor of a, b and c, and let q_1, q_2 be the two children of p. As neither $\{a, b, c\} \in V(q_1)$ nor $\{a, b, c\} \in V(q_2)$, let us assume without loss of generality that $a \in V(q_1)$ and $b, c \in V(q_2)$. It holds that $\{b, c\} \in E(q_2)$, because there is an edge between b and c and they are both descendants of q_2. The node p has to be a join node as $\{a, b\}, \{a, c\} \in E$ and p is by definition the least common ancestor. By definition it holds that $\{\{x, y, z\} \mid x \in V(q_1) \wedge \{y, z\} \in E(q_2)\} \subseteq T(p)$. It follows that $\{a, b, c\} \in T(p)$. Note that $\{a, b, c\}$ is only computed once in the least common ancestor node p and then passed to the parent node. Therefore $T(r)$ only contains $\{a, b, c\}$ once and thus listing all triangles in $T(r)$ solves TRIANGLE ENUMERATION.

We will now analyse the running time. There are n leaf-nodes in the cotree each of which require a constant amount of time to compute. There are at most $n - 1$ union nodes each of which only require a constant amount of time as they only need to point on their children's values. There are at most $n - 1$ join nodes. Each edge and triangle is only added once and all other values do not need to be recomputed. A pointer to the edges and triangles in the children nodes is enough and only require a constant amount of time to be set. The overall running time of this algorithm is therefore in $O(\#\text{T} + n + m)$. □

A similar algorithm as shown in the proof above in which we store the size of the sets rather than the whole sets can be used to solve TRIANGLE COUNTING more efficiently. This is because multiplications and additions of integers can be done significantly faster than performing unions or intersections of sets.

Observation 3.23. Triangle Counting *parametrised by deletion set K to chordal graphs or cographs is solvable in $O(n + m \cdot |K|)$ time.*

The perfect elimination order can be used to solve c-Clique on chordal graphs in linear time. We can exploit the order even more to solve c-Clique parametrised by deletion set K to chordal graphs in $O(n \cdot c^2 \cdot |K|^{c-1} + m)$ time by checking for each $M \subseteq K$ with $|M| < c$ in $O(n \cdot c^2)$ time if there are $c - |M|$ vertices $B \subseteq (V \setminus K)$ such that $M \cup B$ is a clique of size c.

Corollary 3.24. c-Clique *parametrised by deletion set K to chordal graphs is solvable in $O(n \cdot c^2 \cdot |K|^{c-1} + m)$ time.*

Proof. Let K be a set of vertices such that $G' = G - K$ is a chordal graph. Compute a perfect elimination order $\leq_p$ and an ordering $v_1, \ldots, v_n$ of the vertices in $O(n + m)$ time such that $v_i \leq_p v_{i+1}$ holds for all $i \in [0, n-1]$. We can check in $O(m)$ time whether there is a clique of size at least c in G' by checking for each vertex if it has $c-1$ neighbours in G' which come later in the perfect elimination order. By trying all possible combinations we may assume that we know which vertices in K are in a clique C of size c, if there is any, and even the vertex $v \in (V \setminus K) \cap C$ which is the first in the perfect elimination order. We therefore can ignore all vertices w with $w \leq_p v$. We can then check for each $u \in N(v)$ in $O(c)$ time whether it is adjacent to all vertices in $K \cap C$. If it is, then it is also part of C as $N(v)$ is a clique. As $|N(v)| < c$ we can check all such u in $O(c^2)$ time. There are at most k^c possibilities for vertices in $K \cap C$ and n possibilities for v. This algorithm takes $O(n \cdot c^2 \cdot k^c + m)$ time. Note that if we chose k^c vertices in $K \cap C$, then there is no need to check for vertices in $V \setminus K$ which reduces the number of possibilities to $n \cdot k^{c-1}$. $\square$

3.6 Algorithms with Combined Parameters

Green and Bader [34] stated that Triangle Counting parametrised by some vertex cover V' and the maximum degree in this vertex cover $d_{\max} = \max(\{\deg(v) \mid v \in V'\})$ can be solved in $O(|V'| \cdot d_{\max}^2 + n)$ time. We will design an algorithm which solves Triangle Enumeration parametrised by a deletion set K to d-degenerate graphs and the maximum degree $\Delta_K = \max(\{\deg(v) \mid v \in K\})$ in this set K. This algorithm takes $O(|K| \cdot \Delta_K^2 + n \cdot d^2)$ time. Bear in mind that for each vertex cover V' it holds that $G - V'$ is 0-degenerate and hence $O(|K| \cdot \Delta_K^2 + n \cdot d^2) = O(|K| \cdot \Delta_K^2 + n)$. Our result therefore implies the result by Green and Bader. Moreover, keep in mind that for any constant d it holds that $O(|K| \cdot \Delta_K^2 + n \cdot d^2) = O(|K| \cdot \Delta_K^2 + n) \subseteq O(|K| \cdot \Delta^2 + n)$ and thus $O(|K| \cdot \Delta^2 + n)$ is an upper bound for Triangle Enumeration parametrised by maximum degree and feedback vertex set (deletion set to 1-degenerate graphs) or deletion set to planar graphs (deletion set to 5-degenerate graphs).

Theorem 3.25. Triangle Enumeration *parametrised by deletion set K to d-degenerate graphs and the maximum degree Δ_K in K is solvable in $O(|K| \cdot \Delta_K^2 + n \cdot d^2)$ time.*

Proof. Let K be a set such that $G - K$ is d-degenerate and let Δ_K be the maximum degree in K. Formally, $\Delta_K = \max(\{\deg(v) \mid v \in K\})$.

We will show how to list all triangles in G in $O(|K| \cdot \Delta_K^2 + n \cdot d^2)$ time. First, list all triangles which do not contain any vertices in K by applying Algorithm 4 on $G - K$. This algorithm solves Triangle Enumeration parametrised by degeneracy k in $O(n \cdot k^2)$ time as shown in Theorem 3.8. By definition $G - K$ is d-degenerate and thus all triangles with no vertex in K can be listed in $O(n \cdot d^2)$ time.

Next we will prove that all triangles with at least one vertex in K can be listed in $O(|K| \cdot \Delta_K^2)$ time in a second step. Fix an arbitrary order $\leq_a$ of the vertices in K in $O(n + m)$ time. For each $u \in K$ and all possible combinations of two of its neighbours v, w check whether they form a triangle in constant time. If so, then check whether $v \leq_a u \leq_a w$ holds. In doing so, all vertices not in K can be ignored. If this condition holds too, then list $\{u, v, w\}$ as a new triangle. As there are $|K|$ vertices in K and each of which has at most Δ_K neighbours, there are at most $|K| \cdot \Delta_K^2$ possibilities. Hence, all triangles with at least one vertex in K can be enumerated in $O(k \cdot \Delta_K^2)$ time.

We will now prove that all triangles are listed exactly once. If a triangle contains no vertex in K, then it is listed once in the first step as shown in Theorem 3.8. Since we only iterate over all vertices in K in the second step, every triangle found in the second step contains at least one vertex in K and therefore no triangle without any vertex in K is listed more than once. If a triangle contains exactly one vertex in K, then it is only found once in the second step by a similar argument. If a triangle contains more than one vertex in K, then this triangle is found multiple times but only listed once. This is guaranteed by the condition $v \leq_a u \leq_a w$. □

A straight-forward generalisation of the argument above combined with Corollary 3.9 yields the following.

Corollary 3.26. *c-Clique parametrised by deletion set K to d-degenerate graphs and the maximum degree Δ_K in K is solvable in $O(|K| \cdot \Delta_K^{c-1} \cdot c^2 + n \cdot d^{c-1} \cdot c^2)$ time.*

Proof. Let K be a set such that $G - K$ is d-degenerate and let $\Delta_K = \max(\{\deg(v) \mid v \in K\})$ be the maximum degree in K. We make a case distinction whether there is a clique of size c in $G - K$. If so, then by Corollary 3.9 this clique can be found in $O(n \cdot d^{c-1} \cdot c^2)$ time. Otherwise any clique of size at least c has to contain at least one vertex in K. Therefore any clique of size c can be found in $O(k \cdot \Delta_K^{c-1} \cdot c^2)$ time by checking for each vertex $b \in K$ and each possible combination of $c - 1$ of its neighbours $M \subset V$ whether $M \cup \{b\}$ is a clique of size c in $O(c^2)$ time. □

4 Kernelisation Results

Preprocessing is a commonly used technique for computationally hard problems. Generally speaking, reducing a large instance of a hard, let us say NP-hard, problem to a smaller instance in relatively short time and solving this smaller instance by a brute force algorithm often reduces the overall running time significantly. In parametrised complexity theory, this technique is known as kernelisation and is one of the most powerful tools to work with. This technique is by no means restricted to computationally hard problems even though it was invented to tackle problems for which no polynomial-time algorithms are known.

To provide a formal basis for this section we give a definition of kernelisation first. Since we do not limit ourselves to integers as parameters but do allow for example sets, we use the notion $||k||$ to refer to an integer value associated with the parameter k. If k is an integer, then $||k|| = k$ holds. If k is a set, then $||k|| = |k|$ holds.

Definition 4.1. Let P be some parametrised decision problem. Let $\mathcal{A}$ be an algorithm which computes for any input instance (I, k) of P an instance (I', k') in $r(|I|, ||k||)$ time for some polynomial r. We call $\mathcal{A}$ a *kernelisation* for P if

1. $(I, k) \in P \iff (I', k') \in P$ and
2. $|(I', k')| \le f(||k||)$ for some computable function f.

We will call the first condition the correctness and f the size of the kernel (I', k'). If f is some constant, linear, polynomial or exponential function, then we will call (I', k') a constant-size, linear-size, polynomial-size or exponential-size kernel, respectively.

Observe that Triangle Detection and Triangle Counting can be solved in polynomial time. Therefore a constant-size kernel for any parametrisation can be given by simply solving the problem and outputting a trivial yes- or no-instance accordingly. Thus, we are only interested in kernels if there is no known algorithm solving P in $r(|I|,\ ||k||)$ time.

Note that Triangle Enumeration is not a decision problem but specifically asks for all triangles in the graph. Thus, the notion of kernelisation has to be further adapted but this would go beyond the scope of this thesis and is therefore omitted. For an introduction to enumeration kernels see for example Creignou et al. [15].

In this section we will describe several kernelisation results. First we will focus on polynomial-size kernels. Then we will use a technique called partition refinement to prove a theorem that can be used to construct exponential-size kernels for Vertex-Weighted Triangle Counting with any parametrisation that fulfils a certain condition. Then we will prove that this condition is met by the parametrisation deletion set to d-degenerate graphs. Lastly, we will present a kernelisation for Triangle Counting parametrised by maximum degree and deletion set to d-degenerate graphs.

4.1 Polynomial-Size Kernels

We will prove a polynomial-size kernel for Triangle Detection parametrised by maximum independent set size. Recall that an independent set is a set of vertices such that all vertices in this set are pairwise non-adjacent. Furthermore, recall that the maximum independent set size is the largest number k such that there is an independent set of size k in G. From a practical point of view it seems unpromising to use the maximum independent set size as a parametrisation for a polynomial-time problem as computing the parameter is $\mathcal{NP}$-hard and even hard to approximate. However, the maximum independent set size is directly above the minimum dominating set size in the parameter hierarchy. We will prove that there is no algorithm that uses the minimum dominating set size as parametrisation to solve Triangle Detection faster than without a parametrisation. Thus, not only can we draw a line between tractability and intractability, but we can even give a kernelisation for the first parametrisation "above" the line. The idea behind this result is as follows. Lemma 3.4 states that if $k < \sqrt{n}$, then G contains a triangle. Therefore, if $k < \sqrt{n}$, then one can return a simple triangle, and otherwise one can return G.

Proposition 4.2. *There is a kernel with at most $k^2 + 3$ vertices and at most $k^4 + 3$ edges for* Triangle Detection *parametrised by maximum independent set size k which can be computed in constant time.*

Proof. Let $G' = (\{x, y, z\}, \{\{x, y\}, \{x, z\}, \{y, z\}\})$ be a simple triangle and let k be the maximum independent set size of the input graph G. If $k < \sqrt{n}$, then return $(G', 1)$ as a kernel. Otherwise return (G, k) as a kernel. We will prove that this algorithms satisfies both correctness and the specified size.

1. Lemma 3.4 shows that if $k < \sqrt{n}$, then G contains a triangle and thus a simple triangle is a correct kernelisation. It holds trivially that G contains a triangle if and only if G contains a triangle. Thus, the kernelisation is correct. The maximum independent set size of a triangle is one and the maximum independent set size of G is by definition k.

2. A simple triangle contains three vertices and three edges. The graph G has n vertices and at most n^2 edges. Thus, if $k \geq \sqrt{n}$, then it holds that $k^2 \geq n$ and hence $|G| \leq \ k^4 + \ k^2 + \ 6$. Since G has maximum independent set size k and any vertex in a simple triangle is a maximum independent set, it holds that the maximum independent set size k' of the kernel graph I is at most $\max(k, 1)$ and thus $|(I, k')| \leq k^4 + k^2 + 6 + \langle k + 1\rangle = k^4 + k^2 + 6 + \lceil\log(k + 1)\rceil$.

This case distinction can be computed in constant time. Both a simple triangle and returning G can be computed in constant time, too. □

Note that this algorithm requires that G can be returned in constant time and that n and k can be computed in constant time. If these requirements are not met, then one has to spend $O(n + m)$ time to read the input, compute n and k and return a simple triangle or G.

Next we will focus on Triangle Counting parametrised by feedback edge number. Recall that the feedback edge number is the size of the smallest set K of edges such that $G' = (V, E \setminus K)$ is a forest and that this K can be computed in linear time. We use the fact that every forest is triangle-free and hence any triangle has to contain at least one edge in the feedback edge set. Furthermore, we use the fact that for any forest an arbitrarily rooted forest can be computed in linear time in order to compute a linear-size kernel in linear time.

Theorem 4.3. *There is a kernel with at most $4k$ vertices and $5k-1$ edges for* Triangle Counting *parametrised by feedback edge number k which can be computed in $O(n+m)$ time.*

Proof. Let K be a feedback edge set with $|K| = k$. Let $G' = (V, E \backslash K)$ be an arbitrarily rooted forest and let $p(v)$ denote the parent of v in it. We will show how to construct the kernel (I, k').

For every $e = \{v_1, v_2\} \in K$ take the vertices $v_1, v_2, p(v_1), p(v_2)$ into the kernel graph I. Take all edges induced by these vertices into I. Compute the feedback edge number k' of I in $O(n+m)$ time. We show that this satisfies correctness and the specified size.

1. Since G' is a forest and hence triangle-free, each triangle $\{v_x, v_y, v_z\}$ has to contain at least one edge $(v_x, v_y) \in K$. The vertex v_z is the parent of v_x or v_y, the child of one of them or all edges between these three vertices are in K. In the first case, v_z is by construction in the kernel. In the second case, v_z cannot be the child of both v_x and v_y. Let us assume without loss of generality that v_z is the child of v_x. In this case v_z is either the parent of v_y or it holds that $\{v_y, v_z\} \in K$ as v_z cannot be the child of v_y. Either way, the vertex v_z is by construction in the kernel. In the third case, it holds that $\{v_x, v_y\} \in K$ and thus by construction v_z is in the kernel. As v_z is in the kernel in all possible cases and all induced edges are also taken into the kernel, it holds that the triangle $\{v_x, v_y, v_z\}$ is also to be found in the kernel.

2. We take at most four vertices per edge in K into the kernel. Since G' is a forest there are at most $4k-1$ edges from $E \setminus K$ in the kernel. Hence, there are at most $4k$ vertices and $5k-1$ edges in I and thus it holds that $|I| \leq 9 \cdot k - 1$. Since K is a feedback edge set for I as well, it holds that I has a feedback edge number of at most k and thus $|(I, k')| \leq 9 \cdot k - 1 + \lceil \log(k) \rceil$.

We will now prove that this construction can be computed in $O(n+m)$ time. Recall that K and k' can be computed in $O(n+m)$ time. The graph G' can be computed in $O(n+m)$ time by computing $E \setminus K$ in $O(m)$ time. The function p which defines the rooted forest can be computed in $O(n+m)$ time using breadth-first search. It remains to prove that the kernel itself can be computed in $O(n+m)$ time. Consider the following algorithm. Iterate over all edges in K and store all vertices which are an endpoint of one of these edges in a set A. In a second step, iterate over all vertices v in A and store v and $p(v)$ in a new set B. Remember that the function p can be computed in $O(n+m)$ time and each lookup of $p(v)$ takes only constant time afterwards. Finally,

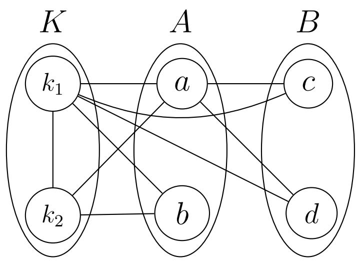

Figure 8: An example of a partition refinement for a set K. The sets A and B form the partition of $V \setminus K = \{a, b, c, d\}$. All vertices within A (within B) have the same neighbours in K.

iterate over all edges and store all edges whose both endpoints are in B in a third set C. These steps can each be computed in $O(n + m)$ time and since $G' = (B, C)$ we can conclude that this procedure computes the specified kernel and thus the given running time is correct. □

4.2 A Generic Approach to Exponential-Size Kernels

In this subsection we will prove the key theorem for the kernelisation results in the next subsection. To this end, we use a technique called partition refinement in order to compute, for some set $K \subseteq V$ of vertices, a partition of the vertices in $V \setminus K$ according to their neighbours in K. That is, we compute a set $\mathcal{P}$ of sets of vertices, such that for each part $P \in \mathcal{P}$ of the partition and any two vertices $a, b \in P$ it holds that $N(a) \cap K = N(b) \cap K$. An example can be seen in Figure 8.

We will use this set $\mathcal{P}$ to construct a kernel for Vertex-Weighted Triangle Counting parametrised by deletion set to d-degenerate graphs in the following subsection. Recall that Vertex-Weighted Triangle Counting has as input a graph $G = (V, E)$, a weight function $w\colon V \to \mathbb{Q}^+$, which assigns a weight to each vertex $v \in V$, and a number $t \in \mathbb{Q}^+$. The weight of a triangle $\{x, y, z\}$ is defined as $w(x) \cdot w(y) \cdot w(z)$ and the question is whether the sum of the weights over all triangles is at least t.

Let (G, w, t) be an instance of Vertex-Weighted Triangle Counting and let $K \subseteq V$ be a set of vertices such that all triangles with at most one vertex in K can be found in some time x which is upper-bounded by some polynomial in the input size and $|K|$. The theorem states that we can construct a kernel (G', w', t') with at most $|K| + 2^{|K|}$ vertices for Vertex-Weighted Triangle Counting parametrised by K in $O(x + n + m)$ time.

The kernelisation consists of two phases. In the first phase, we set $t' = t$ initially and search for all triangles with at most one vertex in K. For each triangle found we reduce t' by the weight of that triangle. We then delete all edges which have no endpoint in K as these edges cannot be part of any further triangles. In the second phase, we compute the partition $\mathcal{P}$ of the vertices in $V \setminus K$ according to K and put only the vertices in K and one vertex for each $P \in \mathcal{P}$. For each vertex $v \in K$ set $w'(v) = w(v)$

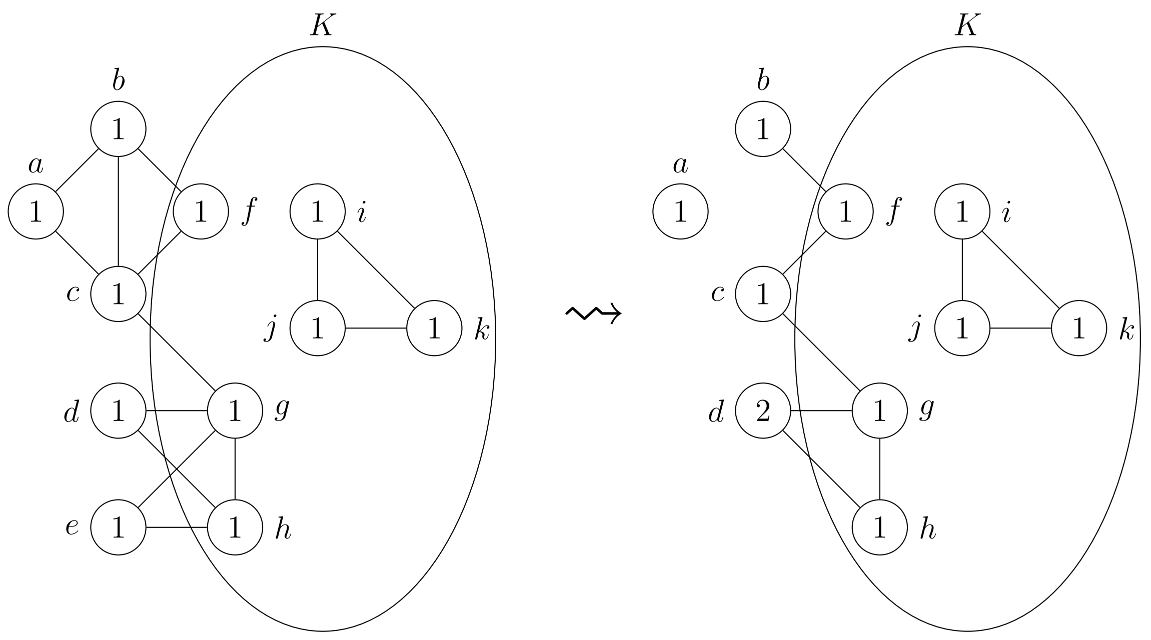

Figure 9: Example of a graph before and after applying the construction of Theorem 4.4

and for each $P \in \mathcal{P}$ put one vertex u in the kernel and set $w'(u) = \sum_{x \in P} w(x)$. An example can be seen in Figure 9. Since there are at most $2^{|K|}$ possible combinations of neighbourhoods in K, there are at most $2^{|K|}$ vertices outside of K left. This leads to the following.

Theorem 4.4. *Let $K \subseteq V$ be a set of vertices such that all triangles with at most one vertex in K can be found in $O(x)$ time where x is upper-bounded by some polynomial in the input size and $|K|$. Then* Vertex-Weighted Triangle Counting *parametrised by K admits a kernel with at most $|K| + 2^{|K|}$ vertices which can be computed in $O(x + n + m)$ time.*

Proof. Recall that Vertex-Weighted Triangle Counting has as input a graph G, a weight function w, which assigns a weight to each vertex, and a number t. The weight of a triangle $\{x, y, z\}$ is defined as $w(x) \cdot w(y) \cdot w(z)$ and the question is whether the sum of the weights over all triangles is at least t. Let (G, w, t) be an input instance of Vertex-Weighted Triangle Counting and let K be a set of vertices such that all triangles with at most one vertex in K can be found in $O(x)$ time where x is upper-bounded by some polynomial in $|(G, w, t)|$ and $|K|$. We construct the kernel $((G', w', t'), K')$ in two phases. In the first phase, we find all triangles containing at most one vertex in K in $O(x)$ time, sum up their weights, store this sum in a variable b and set $t' = t - b$. We then delete all edges which have no endpoint in K in $O(n + m)$ time as those edges are not part of any further triangles. In the second phase, we compute a partition of the vertices in $V \setminus K$ in $O(n + m)$ time using partition refinement as follows.

Compute the set $V \setminus K$ in $O(n+m)$ time. Initialise $\mathcal{P}_0 = \{V \setminus K\}$ and fix an arbitrary order in K. Iterate over all vertices $v_i \in K$ and compute

$$\mathcal{P}_{i+1} = \{P \cap N(v_i) \mid P \in \mathcal{P}_i\} \cup \{P \setminus N(v_i) \mid P \in \mathcal{P}_i\}.$$

That is $\mathcal{P}_{|K|}$ is a partition of the vertices in $V \setminus K$ such that all vertices in one part have the same neighbours in K. This takes $O(k+m)$ time, since there are k iterations, each iteration takes $O(|N(v_i)|)$ time and the sum over all $|N(v)|$ is in $O(m)$. For a more in-depth introduction on partition refinement or the running time analysis see for example Habib and Paul [37]. Note that their algorithm Refine$(\mathcal{P}, N(v))$ is the same algorithm described above for each iteration. Hence Habib and Paul [37, Lemma 10] proved that each iteration takes $O(|N(v_i)|)$ time.

We will now prove that this procedure indeed computes a partition of the vertices in $V \setminus K$ such that all vertices in one part have the same neighbours in K. First, we will prove that each vertex $w \in V \setminus K$ is in exactly one part of the partition. Since we initialise $P = \{V \setminus K\}$, each vertex in $V \setminus K$ is in exactly one part in the beginning. In each iteration, each vertex $w \in V \setminus K$ is in exactly one part $P \in \mathcal{P}$ and it is either in $N(v_i)$ or it is not. Either way, it is either in $P \cap N(v_i)$ or it is in $P \setminus N(v_i)$ and it is not contained in any other part as each of this operations only divides parts and never inserts vertices. Next, we will show that in the end all vertices in one part of the partition have the same neighbours in K. Assume towards a contradiction that there is a part $P \in \mathcal{P}$, vertices $x, y \in P$ and $k \in K$ such that $k \in N(x)$ and $k \notin N(y)$. In one iteration all parts are divided by $N(k)$ and thus x, y are in different parts after this iteration. Since all following iterations only divide the parts further and do not merge any parts, x and y cannot be in the same partition. This is a contradiction to the assumption and thus proves the claim. Moreover, we will show that any two vertices $x, y \in V \setminus K$ which have the same neighbours in K are contained in the same part of the partition after the last iteration. By construction, x and y are contained in the same part after the initialisation. In each iteration vertices of the same part are only separated if one of them has the current $v_i \in K$ as a neighbour and the other does not. Since x and y have exactly the same neighbours in K, they are never separated and thus they are contained in the same part of the partition after the last iteration.

The kernel graph contains of two kinds of vertices. All vertices in K and one representative vertex u for each non-empty part $P \in \mathcal{P}$. For each vertex $v \in K$ set $w'(v) = w(v)$ and for each part P delete all but one vertex $u \in P$ and set $w'(u) = \sum_{x \in P} w(x)$. Let V' be the vertex set of G' and let $w_0 = \max_{v \in V'}(w'(v))$ be the maximum weight of a vertex in G'. Set $t' = t'/w_0^3$, iterate over all vertices $v \in V'$ and set $w'(v) = w'(v)/w_0$. This limits the new maximum weight w_0 to one and since t' is adjusted appropriately this does not effect the correctness of the kernelisation.

We will prove that this kernelisation satisfies correctness, that the kernel graph G' contains at most $|K| + 2^{|K|}$ vertices and that there is a function f such that $|((G', w', t'), K)| \leq f(|K|)$.

1. Any triangle in the graph G either contains at most one, exactly two or exactly three vertices in K. In the first case, the triangle is, by assumption, found in

the first phase. In the second case, two vertices u, v in the triangle are in K and the third vertex w is in one part B of the partition. By construction, one representative vertex $s \in B$ is contained in G'. As shown above, it holds that w and s have the same neighbours in K and thus $\{u, v, s\}$ is a triangle in G. As all edges between s, u and v have at least one endpoint in K, the triangle $\{u, v, s\}$ is also in G'. The weight of s in G' equals the sum of the weights of all vertices $b \in B$ in G'. Since all vertices in B have the same neighbours in K, the weight sum of triangles in G containing u, v and a vertex in B is $w(u) \cdot w(v) \cdot (\sum_{b \in B} w(b))$. This is exactly the weight of $\{u, v, s\}$ in G'. In the last case all vertices are contained in K and thus the triangle is also in G' with exactly the same weight. As all vertices and edges in G' are also in G we do not construct any new triangles and thus the kernelisation is correct.

2. We put all $|K|$ vertices in K in G' and one out of every non-empty part of the partition. Since there are at most $2^{|K|}$ possible combinations of neighbours in K, there are at most $2^{|K|}$ partitions which contain vertices in $V \setminus K$ and thus G' contains at most $|K| + 2^{|K|}$ vertices.

3. There are at most $(|K|+2^{|K|})^2$ edges in G' as there are at most $|K|+2^{|K|}$ vertices in G'. Let V' be the set of vertices in G'. If $t' > (|K|+2^{|K|})^3$, then we can return a trivial no-instance as each triangle has weight at most one and there are at most $(|K| + 2^{|K|})^3$ triangles. Otherwise it holds that $\langle t' \rangle \leq \lceil \log((|K| + 2^{|K|})^3) \rceil$. Since w' is defined over all vertices in G' and $w_0 = 1$ holds, it holds that $\langle w' \rangle \leq |K| + 2^{|K|}$. Finally, K is the parameter for the kernel as well and thus it holds that

$$|((G', w', t'), K')| \leq 3 \cdot |K| + 2^{|K|+1} + (|K| + 2^{|K|})^2 + \lceil \log((|K| + 2^{|K|})^3) \rceil.$$

Thus, there is a function f such that $|((G', w', t'), K')| \leq f(|K|)$.

This construction takes $O(x + n + m)$ time in the first phase and $O(n + m)$ time in the second phase. This results in a total running time in $O(x + n + m)$ which is upper-bounded by some polynomial r in $|(G, w, t)|$ and $|K|$. □

Note that every time we use Theorem 4.4 to construct a kernel we can immediately conclude a kernel for Triangle Detection parametrised by the same parameter. This is due to the fact that if we find any triangle in the first phase of the construction above, then we can return a simple triangle. In the second phase we do not have to assign weights to any vertices as finding one triangle is enough for Triangle Detection. We therefore omit kernels for Triangle Detection which use Theorem 4.4.

4.3 Exponential-Size Kernel

In this section we will use Theorem 4.4 to prove an exponential-size kernel for Vertex-Weighted Triangle Counting parametrised by deletion set to d-degenerate graphs. This implies exponential-size kernels for Vertex-Weighted Triangle

parameter set	d
vertex cover	0
feedback vertex set	1
distance to outerplanar graphs	2
distance to planar graphs	5
distance to d-degenerate graphs	d

Figure 10: Degeneracy d of $G - K$ where K is the respective parameter set.

Counting parametrised by deletion set to planar graphs, deletion set to outerplanar graphs, feedback vertex set and vertex cover. See Figure 10 for more information on the value of d for the different parametrisations.

We will prove that for any set K of vertices such that $G - K$ is d-degenerate, all triangles with at most one vertex in K can be found in $O(n \cdot d \cdot (|K| + d))$ time. Since $G - K$ is d-degenerate, all triangles with no vertex in K can be found in $O(n \cdot d^2)$ time as shown in Theorem 3.8. All triangles with exactly one vertex in K can be found in $O(n \cdot d \cdot k + m)$ time by the following algorithm. Compute the degeneracy order $\leq_d$ in $O(n + m)$ time. Try for every $v \in K, w \in V \setminus K$ and every $u \in N(w) \setminus K$ with $w \leq_d u$, if $\{u, v, w\}$ is a triangle in constant time. Note that there are at most d of such u vertices for every w because $G - K$ is d-degenerate. Furthermore, note that since $G - K$ is d-degenerate and degeneracy upper-bounds average degree, there are at most $n \cdot d$ edges which have no endpoint in K. There are at most $n \cdot k$ edges with an endpoint in K and thus the number m of edges in G is in $O(n \cdot (k + d))$. For any constant d, all triangles with exactly one vertex in K can be found $O(n \cdot k)$ time. This observation and Theorem 4.4 yield the following.

Lemma 4.5. *Let K be a set such that $G - K$ is d-degenerate. All triangles with at most one vertex in K can be found in $O(n \cdot d \cdot (|K| + d))$ time.*

Proof. Let K be a set of vertices such that $G - K$ is d-degenerate. We will show that all triangles with at most one vertex in K can be found in $O(n \cdot d \cdot (|K| + d))$ time.

First, we compute the degeneracy ordering of $G - K$, that is, an ordering of the vertices such that every vertex has at most d neighbours which come later in the ordering. Recall that this ordering always exists, because $G - K$ is d-degenerate, and can be computed in $O(n + m)$ time.

We will iteratively find all triangles which contain the first vertex in the degeneracy order and at most one vertex in K and then delete this vertex. Therefore every vertex in $V \setminus K$ is checked at some point and we can conclude that all triangles with at most one vertex in K are found. Finding all those triangles can be done as follows.

Let v be the first vertex in the degeneracy order. For every $u \in N(v) \setminus K$ and every $w \in N(v)$ check if $\{u, v, w\}$ is a triangle in constant time. Delete v afterwards and continue with the next iteration. Note that there are n iterations and in each iteration there are at most d possibilities for u and at most $|K| + d$ possibilities for w. This results in a total of at most $n \cdot d \cdot (|K| + d)$ checks which each can be computed in constant time. Furthermore, note that every vertex and every edge is deleted at most once and that m is upper-bounded by $n \cdot (|K| + d)$.

The whole construction therefore takes $O(n \cdot d \cdot (k + d))$ time. □

From Lemma 4.5 and Theorem 4.4 it follows directly that there is a kernel with at most $|K| + 2^{|K|}$ vertices for VERTEX-WEIGHTED TRIANGLE COUNTING parametrised by deletion set K to d-degenerate graphs which can be computed in $O(n \cdot d \cdot (|K| + d))$ time.

Theorem 4.6. *There is a kernel with at most $|K| + 2^{|K|}$ vertices for* VERTEX-WEIGHTED TRIANGLE COUNTING *parametrised by deletion set K to d-degenerate graphs which can be computed in $O(n \cdot d \cdot (|K| + d))$ time.*

4.4 Combined Parameters

In this subsection we will describe a kernelisation for TRIANGLE COUNTING parametrised by maximum degree Δ and deletion set K to d-degenerate graphs.

As a special case we want to point to the parametrisation vertex cover, which is the same as deletion set to 0-degenerate graphs. Except for isolated vertices $|G|$ is upper-bounded by $|K| \cdot \Delta$.

Observation 4.7. *There is a kernel with at most $|K| \cdot (\Delta + 1)$ vertices and at most $|K| \cdot \Delta$ edges for* TRIANGLE COUNTING *parametrised by maximum degree Δ and vertex cover K which can be computed in $O(n + m)$ time.*

Proof. Let K be some vertex cover. We will construct a kernel (G', t') from input (G, t). We will then show that $|(G', t')|$ is upper-bounded by $O(|K| \cdot \Delta)$ and prove correctness.

First, iterate over all vertices and put all vertices which have at least one neighbour in the kernel graph G' and count the number n' of vertices in G'. Put all edges in G into G'. If $t > \binom{n'}{3}$, then we can return a trivial no-instance as there are at most $\binom{n'}{3}$ triangles in G. Otherwise, $\langle t \rangle$ is upper-bounded by $\lceil \log(|G'|^3) \rceil$ as $|G'|$ upper-bounds n' and t can be represented using $\lceil \log(t) \rceil$ bits. We will now prove that $|G'|$ is upper-bounded by $O(|K| \cdot \Delta)$.

Any edge has at least one endpoint in any vertex cover. Hence, G' contains at most $|K| \cdot \Delta$ edges. Since there are no degree-zero vertices in G', it follows that there are at most $|K| \cdot \Delta$ vertices not in K and thus at most $|K| \cdot (\Delta + 1)$ vertices in G'.

Note that $O(x)$ upper-bounds $\log(x^3)$ and thus the kernel described above is of size $O(|K| \cdot \Delta + \log((|K| \cdot \Delta)^3)) = O(|K| \cdot \Delta)$. Since G and G' only differ in their isolated vertices, K is a vertex cover for G', too. Therefore it holds that the kernel $((G', t'), K)$ is of size $O(|K| \cdot \Delta)$ and that there are exactly the same triangles in G and G'.

The construction above takes $O(n + m)$ time as we only iterate over all vertices once and then copy all edges once. □

Next we will prove that there is a kernel of size $O(k \cdot \Delta)$ for TRIANGLE COUNTING parametrised by deletion set K to d-degenerate graphs and maximum degree Δ. The main idea is to count all triangles with at most one triangle in K in $O(n \cdot d \cdot (k + d))$

time using Lemma 4.5 and only put K and all neighbours of vertices in K in the kernel. This yields the following.

Theorem 4.8. *There is a kernel of size $O(|K| \cdot \Delta)$ for* Triangle Counting *parametrised by deletion set K to d-degenerate graphs and maximum degree Δ which can be computed in $O(n \cdot d \cdot (|K| + d))$ time.*

Proof. Lemma 4.5 shows that all triangles with at most one vertex in K can be found in $O(n \cdot d \cdot (|K| + d))$ time. Note that there has to be an edge between all vertices in a triangle and therefore one has to check only all neighbours of vertices in K in the last part of the proof of Lemma 4.5. Therefore the running time is in $O(n \cdot d^2 + |K| \cdot \Delta \cdot d)$. It remains to show that there is a kernel (G', t') of size at most $O(|K| \cdot \Delta)$ which contains all triangles with at least two vertices in K and which can be computed in $O(n + m)$ time.

We set $t' = t - b$ accordingly to the number b of triangles found so far. We then construct a graph which contains as many triangles as there are triangles in G with at least two vertices in K.

First delete all edges in G which have no endpoint in K in $O(n + m)$ time. All triangles left therefore have at least two vertices in K because each vertex in K can be an endpoint of at most two edges in a triangle and each triangle contains three edges. Next, delete all vertices which are neither in K nor have a neighbour in K in $O(n+m)$ time by first computing all neighbours of vertices in K in $O(n + m)$ time and then deleting all other vertices in $O(n + m)$ time. There are $|K|$ vertices in K left and at most $|K| \cdot \Delta$ vertices not in K. If $t' > (|K| \cdot (\Delta + 1))^3$, then we can return a simple no-instance. Otherwise, it holds that $\langle t \rangle \leq \lceil \log((|K| \cdot (\Delta + 1))^3) \rceil$ is upper-bounded by $O(|K| \cdot \Delta)$, too.

This construction can be computed in $O(n \cdot d^2 + |K| \cdot \Delta \cdot d)$ time. Note that the degeneracy d is an upper bound for the average degree in $G - K$ and that there are at most $|K| \cdot \Delta$ edges with an endpoint in K. Hence, m is upper-bounded by $n \cdot d + |K| \cdot \Delta$ and thus does not appear in the running time. □

5 Computational Hardness Results

Up to this point we have shown results on what can be done to solve Triangle Detection, Triangle Counting and Triangle Enumeration in some cases more efficiently. To this end, we explored which parametrisations are useful for this task. In this section, we will do quite the opposite. We will show which parametrisations cannot be used to design faster algorithms—at least not in general. By that we mean, that there might still be special algorithms for graphs of constantly bounded parameter size but there is some constant c such that solving the respective problem on graphs with parameter size at least c is as computationally hard as the general case of unbounded parameter size. Even though these results, by definition, cannot be used to implement faster algorithms we believe that they are still valuable. Knowing which parameters should not be used to construct faster algorithms saves time for those who design algorithms and let these people focus on parametrisations which yield fast algorithms.

In order to prove that certain parametrisations are unsuited to design faster algorithms for finding triangles we present a new notion of hardness which we call *general problem hardness*. Generally speaking, the idea is the same as in for example $\mathcal{NP}$-hardness. We start with some assumption, in this case that there is no linear-time algorithm for Triangle Detection, and show that the existence of a fast algorithm for Triangle Detection (or Triangle Counting) with some parametrisation contradicts our assumption. To this end we "reduce" an arbitrary instance of Triangle Detection (or Triangle Counting) to a new instance of the same problem with bounded parameter size such that either both instances are yes-instances or both are no-instances.

We now present a formal definition of the idea above.

Definition 5.1. Let P be some decision problem, let J be some parametrisation, let f be some fixed computable function and let ℓ be some non-negative integer. We call P *ℓ-general problem hard($f(x)$) (ℓ-GP-hard($f(x)$))* with respect to J if there exists an algorithm $\mathcal{A}$ transforming any input instance (I, k) of P parametrised by J to a new instance (I', k') of P parametrised by J where

I) $\mathcal{A}$ has running time in $O(f(|I|))$,

II) $I \in P \iff I' \in P$,

III) $||k'|| \leq \ell$ and

IV) $|I'| \in O(|I|)$.

We call P *general problem hard($f(x)$) (GP-hard($f(x)$))* with respect to J if there exists an ℓ such that P is ℓ-GP-hard($f(x)$) with respect to J. We omit the running time and call P general problem hard (GP-hard) with respect to J if f is a linear function.

We will now show that GP-hardness($f(x)$) implies intractability. To this end, let (I, k) be some input instance of P with some parametrisation J and let g be any computable function. We will prove that if P cannot be solved in $O(f(|I|))$ time and P is ℓ-GP-hard$(f(x))$ with respect to J, then J cannot be used to design an algorithm that takes $O(f(|I|) \cdot g(||k||))$ time. The main idea is to assume that there is an algorithm that uses J to achieve the specified running time and use both this algorithm and the reduction described in Definition 5.1 to construct an $O(f(|I|))$-time algorithm for the unparametrised variant of P. This leads to the following.

Lemma 5.2. *Let P be some decision problem, let J be some parametrisation and let f be some fixed computable function. If there is no algorithm solving each instance I of P in $O(f(|I|))$ time and P is ℓ-GP-hard$(f(x))$ with respect to J, then there is no algorithm solving each instance (I', k) of P parametrised by J in $O(g(||k||) \cdot f(|I'|))$ time for any computable function g.*

Proof. The proof is by contradiction. Assume towards a contradiction that there is an algorithm $\mathcal{B}$ which solves each instance (H, j) of P parametrised by J in $O(g(||j||) \cdot f(|H|))$ time. Let (I, k) be an arbitrary instance of P parametrised by J. Since P is ℓ-GP-hard$(f(x))$ with respect to J, there is an algorithm $\mathcal{A}$ which transforms (I, k) into a new instance (I', k') of P parametrised by J in $O(f(|I|))$ time such that $||k'|| \leq \ell$, $I \in P$ if and only if $I' \in P$ and $|I'| \in O(|I|)$. By assumption, algorithm $\mathcal{B}$ solves (I', k') in $O(g(||k'||) \cdot f(|I'|))$ time. Since $||k'|| \leq \ell$ and $|I'| \in O(|I|)$, it holds that $O(g(||k'||) \cdot f(|I'|)) \subseteq O(f(|I|))$. Since $I \in P$ if and only if $I' \in P$ holds, this construction solves P in $O(f(|I|))$ time which is a contradiction to the assumption that there is no algorithm solving P in $O(f(|I|))$ time. $\square$

Note that ℓ-GP-hardness only states that if the parameter k is at least ℓ, then P parametrised by J is as computationally hard as the unparametrised problem P. If $k \in [0, \ell)$, then there might be an algorithm which uses J to solve the problem more efficiently.

Back to the problem of finding triangles, Williams and Williams [63] conjectured that there is no linear-time algorithm for Triangle Detection. Thus, if Triangle Detection is GP-hard$(n + m)$ with respect to some parametrisation J, then there is no algorithm solving each instance (G, k) of Triangle Detection parametrised by J in $O((n + m) \cdot g(||k||))$ time for any computable function g, unless the conjecture by Williams and Williams is false.

Recall that Triangle Detection is a special case of Triangle Counting and Triangle Enumeration and therefore if Triangle Detection is ℓ-GP-hard with respect to some parametrisation J, then Triangle Counting is ℓ-GP-hard with respect to J, too. This also implies that there is no algorithm solving each instance (I, k) of Triangle Counting or Triangle Enumeration parametrised by J in $O(|I| \cdot g(||k||))$ time for any computable function g, unless there is an algorithm solving Triangle Detection in $O(n + m)$ time. Note that Triangle Enumeration is not a decision problem and hence the notion of GP-hardness has to be further adapted in order to be applicable to Triangle Enumeration, but this would go beyond the scope of this thesis.

In this section we will prove several GP-hardness results. First we will observe that Triangle Detection is 0-GP-hard with respect to minimum degree or bisection width. Then we will prove that Triangle Detection is 4-GP-hard with respect to maximum diameter of components and Triangle Counting is 2-GP-hard with respect to maximum diameter of components. In the last subsection we will prove that Triangle Detection is 3-GP-hard with respect to minimum dominating set and chromatic number. We want to highlight Theorem 5.7 because it functions as a proof of concept. The theorem shows that Triangle Detection is 3-GP-hard with respect to minimum dominating set. We consider this result non-obvious and therefore the notion of general problem hardness can be used to achieve non-trivial results.

5.1 Minimum Degree and Bisection Width

We will show that Triangle Detection is 0-GP-hard with respect to minimum degree and bisection width. Recall that the minimum degree is the minimum number of neighbours over all vertices in G. Adding a single vertex without any neighbours sets the minimum degree in G to zero and does not affect the number of triangles in G.

Observation 5.3. Triangle Detection *is* 0-*GP-hard*(1) *with respect to minimum degree.*

Proof. Add a single vertex without any neighbours. We will show that this construction fulfils all requirements of Definition 5.1.

I) Adding a single vertex can be done in constant time.

II) Adding a single vertex neither adds any triangles nor deletes any.

III) Since this new vertex has no neighbours, the minimum degree of the resulting graph is 0.

IV) Adding a single vertex only increases the size of the graph by a constant amount.

□

Note that all degree-zero and degree-one vertices can be deleted in $O(n)$ time in a preprocessing step as shown in Reduction Rule 2.5. Nonetheless, this only reduces the size of the input and there is no hope for an algorithms solving each instance (G, k) of Triangle Detection parametrised by minimum degree in $O(|G| \cdot g(k))$ time for some computable function g, unless Triangle Detection can be solved in $O(|G|)$ time. More precisely, there is no algorithm solving (G, k) in $O(f(|G|) \cdot g(k))$ time for some $f(|G|) \in O(|G|)$ and some computable function f, unless Triangle Detection can be solved in $f(|G|)$ time.

Next we will show that Triangle Detection is 0-GP-hard to bisection width. Recall that the bisection width of a graph G is the minimum amount of edges one has to delete in G such that the vertices of the resulting graph can be partitioned into two sets A, B such that $||A| - |B|| \leq 1$ and that there is no edge between a vertex in A and a vertex in B.

Observation 5.4. Triangle Detection *is* 0*-GP-hard with respect to bisection width.*

Proof. Add another copy of the graph to the graph. We will show that this construction fulfils all requirements of Definition 5.1.

I) This construction can be done $O(n+m)$ time.

II) Adding another copy does not delete any triangle. Any triangle in the original graph now appears exactly twice in the resulting graph. No other triangles are added and thus there is a triangle in the new graph if and only if there is a triangle in the old graph.

III) The bisection width of the resulting graph is 0 because there is no edge between the two copies of the original graph and both of this copies have the same number of vertices.

IV) This transformation exactly doubles the size of the graph. □

5.2 Maximum Diameter of Components

In this subsection we will prove that Triangle Detection is 4-GP-hard with respect to maximum diameter of components and Triangle Counting is 2-GP-hard with respect to maximum diameter of components. Recall that the maximum diameter of components is the maximum pairwise distance between any two vertices in the same connected component in G.

The key point to prove that Triangle Detection is 4-GP-hard with respect to maximum diameter of components is to observe that one can add paths of three edges between all vertices in G without adding any new triangles. This yields the following.

Proposition 5.5. Triangle Detection *is* 4*-GP-hard*(n) *with respect to maximum diameter of components.*

Proof. Let $V = \{v_1, v_2, \ldots, v_n\}$ be the set of vertices of the input graph. Add a set $U = \{u_1, u_2, \ldots, u_n\}$ of vertices and a set $E' = \{\{v_i, u_i\} \mid i \in [1, n]\}$ of edges to the graph. Add another vertex w to the graph and add edges $\{\{w, u_i\} \mid i \in [1, n]\}$, that is, add an edge between w and each $u \in U$. An example of this construction can be seen in Figure 11. We will prove that this construction satisfies all conditions of Definition 5.1. Let G be the input graph and G' be the constructed graph.

I) The given construction takes $O(n)$ time, because one can add a single vertex w in constant time and then iterate over all vertices, add a new vertex u_i for every vertex v_i in the original graph and add edges $\{v_i, u_i\}$ and $\{u_i, w\}$. Each iteration takes constant time and there are n iterations.

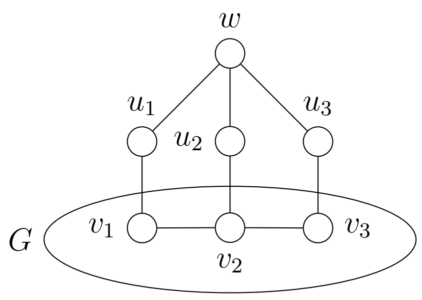

Figure 11: An example of a graph constructed in the proof of Proposition 5.5.

II) We will show that there is a triangle in G' if and only if there is a triangle in G. The set U is independent as each $u \in U$ is only incident to w and exactly one $v \in V$. As w does not have any neighbours outside this independent set, w cannot be part of any triangle. Every $u \in U$ cannot be part of a triangle as they have degree two and one of their neighbours is w. Since the graph induced by all other vertices is exactly G, it holds that G' contains a triangle if and only if G contains a triangle.

III) We will show that the maximum diameter of components in G' is four. The distance between w and any other vertex is at most two as w is connected to all vertices in U and each $v \in V$ is connected to one vertex in U. As each vertex has distance at most two to w, the distance between two arbitrary vertices is at most four.

IV) The graph G' has $2n + 1$ vertices and $m + 2n$ edges. It therefore holds that $|G'| \in O(|G|)$. □

Note that the set U in the proof above is only required to guarantee that no new triangles are constructed. This observation can be used to prove that Triangle Counting is 2-GP-hard(n) with respect to maximum diameter of components.

Proposition 5.6. Triangle Counting *is* 2-*GP-hard*(n) *with respect to maximum diameter of components.*

Proof. Let (G, t) be an arbitrary input instance for Triangle Counting. We will construct a new instance (G', t') which satisfies all conditions of Definition 5.1.

Let $t' = t + m$ and construct G' from G by adding a single vertex w to the graph and connecting it to every $v \in V$ by an edge.

I) The given construction takes $O(n)$ time because w can be added in constant time and an edge between w and a vertex v in G can be added in constant time as well. As there are n of such new edges to be added, adding all of them takes $O(n)$ time.

II) We will prove that G' contains at least t' triangles if and only if G contains at least t triangles. If G contains at least t triangles, then G' contains t triangles which do not contain w because G and G' only differ in the vertex w and all edges which have w as an endpoint. For every edge $\{u, v\}$ in G there is a triangle $\{u, v, w\}$ in G' as all vertices are adjacent to w. Therefore there are at least $t + m = t'$ triangles in G'.

We will now prove that if there are less than t triangles in G, then there are less than t' triangles in G'. Once again, G and G' only differ in the vertex w and hence there are less than t triangles in G' which do not contain w. Since w forms a triangle with any two vertices u, v if and only if $\{u, v\} \in E$ (and $u, v \neq w$), w is contained in exactly m triangles. Thus, G' contains less than $t+m = t'$ triangles.

Thus, G' contains at least t' triangles if and only if G contains at least t triangles.

III) We will prove that the maximum diameter of components in G' is at most two. The distance between w and any other vertex is one as w is connected to all other vertices. As each vertex has distance one to w, the distance between two arbitrary vertices is at most two.

IV) The graph G' has $n+1$ vertices and $m+n$ edges. Thus, it holds that $|G'| \in O(|G|)$. □

5.3 Minimum Dominating Set and Chromatic Number

In this subsection we will prove that Triangle Detection is 3-GP-hard with respect to minimum dominating set and chromatic number. This result is noticeable for two reasons. First, both of these claims can be proven with only one construction even though minimum dominating set and chromatic number are neither bounded by nor considered very similar to one another. Second, a vertex cover and a maximum independent set generally behave somewhat similar to a minimum dominating set but in this case they behave very differently. Both a vertex cover and a maximum independent set can be used as parameters to design linear-time algorithms for Triangle Detection but Triangle Detection parametrised by a constant size minimum dominating set cannot even be solved in $O(n^2)$ time, unless there is an $O(n^2)$-time algorithm solving Triangle Detection without parametrisation.

Theorem 5.7. Triangle Detection *is* 3-*GP-hard*(n) *with respect to minimum dominating set or chromatic number.*

Proof. Let $G = (\{v_1, v_2, \ldots, v_n\}, E)$ be the input graph. Consider the graph $G' = (V', E')$ constructed as follows. Put three copies of the vertices as three independent sets into G'. For each edge $\{v, w\}$ in G add edges between the corresponding vertices such that each copy induces an independent set. Then simply add one vertex for each of the copies and connect it to each vertex in this copy.

Formally, this is $V' = V_1' \cup V_2' \cup V_3'$ with $V_i' = \{v_1^i, v_2^i, \cdots, v_n^i, u_i\}$, $i \in \{1, 2, 3\}$ and

$$E' = \{\{v_x^i, v_y^j\} \mid i, j \in \{1, 2, 3\} \wedge i \neq j \wedge \{v_x, v_y\} \in E\} \cup \{\{u_i, v_j^i\} \mid i \in \{1, 2, 3\} \wedge j \in [1, n]\}.$$

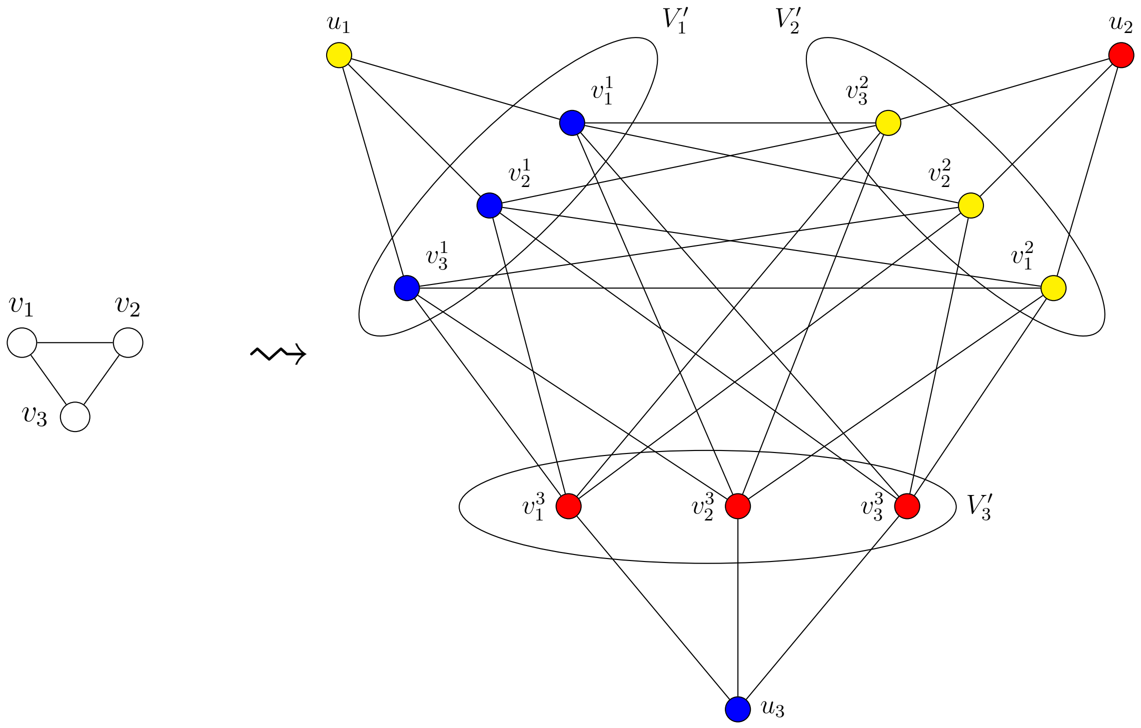

Figure 12: A simple triangle and the resulting graph with minimum dominating set size and chromatic number three.

Consider the very simple example of a single triangle. Figure 12 illustrates both the input graph and the resulting graph of the construction described above.

We will prove that this construction fulfils all requirements of Definition 5.1.

I) The given construction can be done in $O(n + m)$ as one only iterates once over all vertices and edges and adds $3n + 3$ vertices and $6m + 3n$ edges to G'.

II) We will prove that G' contains a triangle if and only if G does. We first show that if G contains a triangle, then G' contains a triangle and then we prove that if G' contains a triangle, then so does G.

Assume that G contains a triangle $\{v_x, v_y, v_z\}$. By construction, G' contains vertices v_x^1, v_y^2 and v_z^3 and edges $\{v_x^1, v_y^2\}, \{v_y^2, v_z^3\}$ and $\{v_x^1, v_z^3\}$. This is a triangle in G'.

Assume that G' contains a triangle. This triangle cannot contain u_i as it is only incident to vertices in V_i' which are independent sets by construction. As V_1', V_2' and V_3' are independent sets, any triangle $\{v_x, v_y, v_z\}$ must contain exactly one vertex of each of them. Let, without loss of generality, $v_x = v_a^1 \in V_1'$, $v_y = v_b^2 \in V_2'$ and $v_z = v_c^3 \in V_3'$. For $i \neq j$, it holds that $\{v_a^i, v_b^j\}, \{v_a^i, v_c^j\}, \{v_b^i, v_c^j\} \in E'$ if and only if $\{v_a, v_b\}, \{v_a, v_c\}, \{v_b, v_c\} \in E$, respectively. Thus, $\{v_a, v_b, v_c\}$ is a triangle in G.

III) Now we will prove that G' has a minimum dominating set of size three and chromatic number three.

As each vertex in $V_i' \setminus \{u_i\}$ is connected to u_i the set $\{u_1, u_2, u_3\}$ is a dominating set for G'. By construction, there is no vertex that is adjacent to two vertices in $\{u_1, u_2, u_3\}$ and hence $\{u_1, u_2, u_3\}$ is a minimum dominating set.

As each V_i' is an independent set we can label all of it's vertices with i. As u_1 is only connected to vertices in V_1' we can label u_1 with 2. Analogously, we can label u_2 with 3 and u_3 with 1. The graph G' has therefore chromatic number three.

IV) The graph G' contains $3n + 3$ vertices and $6m + 3n$ edges. Thus, it holds that $|G'| \in O(|G|)$. □

Since graphs with chromatic number two are bipartite and hence do not contain any triangles, the specified bound for chromatic number is tight even for TRIANGLE COUNTING. The construction in Proposition 5.6 proves that TRIANGLE COUNTING is 1-GP-hard with respect to minimum dominating set. Recall that there is a vertex w that is adjacent to each other vertex in the graph. Minimum dominating set and chromatic number therefore do not always behave identically for other triangle finding problems than TRIANGLE DETECTION.

6 Conclusion

This thesis contributes to the FPT in $\mathcal{P}$ field in two different ways.

On the one side, we explored the parametrised complexity of finding triangles and found some promising results not only for algorithms solving the problems but also kernelisation algorithms. See Section 1.2 for a complete list of our results. From our point of view, algorithms using the degeneracy of the input graph seem the most promising.

On the other side, we introduced the notion of general problem hardness which is to the best of our knowledge the first approach to construct a tool to systematically prove that certain parametrisations do not entail faster algorithms for problems in $\mathcal{P}$. The simple idea behind is to transform an arbitrary input instance of a problem in $\mathcal{P}$ into a new equivalent instance whose parameter size (or value) is bounded by some constant in less time than one needs to solve the instance directly. If this can be done and one assumes, towards a contradiction, that there is an algorithm which uses this parametrisation to solve this new instance sufficiently fast, then one can solve every instance of the problem in less time than is needed to solve the instance which is a contradiction.

Future Work We conclude this thesis with a list of remaining open problems and further research directions.

Encyclopaedia on Parametrisations. No single paper can cover all reasonable graph parametrisations. Hence, a possible direction for future work is to find parametrisations that are not studied in this thesis and which yield fast algorithms for finding triangles in practice. Furthermore, a collection point for all these parametrisations, algorithms to compute and approximate them and the relations between them in a citable manner would be handy to say the least. Some effort in this direction has already been made [19, 41, 57, 59], but each of these works focuses on one of the aspects only.

Implementation. It still remains open whether or not our parametrised algorithms can beat the existing non-parametrised algorithms in practice since this thesis is purely theoretical and thus these algorithms are neither implemented nor empirically tested yet. From our point of view, the algorithms presented in Theorems 3.6, 3.8 and 3.25 using degeneracy and feedback edge number as parametrisations seem the most promising for practical applications.

GP-hardness. The notion of general problem hardness (GP-hardness for short) is universally applicable for all problems in $\mathcal{P}$. It still remains open to see whether this new tool is capable of establishing itself in the field of FPT in $\mathcal{P}$. Hence, it is interesting to see whether general problem hardness can be applied on other problems and whether there is a simple way to adapt the definition such that it can be used on non-decision problems, too.

Enumeration Problems in the field of FPT in $\mathcal{P}$. There are concepts specially designed for enumeration problems. These concepts include enumerative kernels and Delay-FPT algorithms [15]. These concepts are foremost designed for $\mathcal{NP}$-hard problems. Nevertheless, we consider it interesting to see if these concepts are easily applicable to polynomial-time solvable enumeration problems like TRIANGLE ENUMERATION.

Boolean Matrix Multiplication and All-Pairs Shortest Path. Williams and Williams [63] have shown that any truly subcubic-time algorithm that decides whether an edge-weighted graph contains a triangle of negative total edge weight can be used to solve the all-pairs shortest path problem and boolean matrix multiplication in truly subcubic time. We have proven that TRIANGLE ENUMERATION can be solved in parametrised truly subcubic time in special cases, that is, for a fixed constant parameter size, we have given algorithms which run in $O(n^{3-\delta})$ time for some $\delta > 0$. Since it only takes constant time per triangle to check whether it has negative total edge weight, we can construct parametrised truly subcubic-time algorithms to decide whether a graph contains a triangle of negative total edge weight. It remains to implement both the algorithms by Williams and Williams that transform an instance of boolean matrix multiplication or all-pairs shortest path to an instance of negative triangle detection and the parametrised algorithms for negative triangle detection to see if the combination of these two can beat the existing algorithms for boolean matrix multiplication and the all-pairs shortest path problem in practice.

References

[1] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014. 7, 8

[2] Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391. SIAM, 2016. 9

[3] Martin Aigner and Günter M. Ziegler. *Proofs from the Book*. Springer, 2010. 18

[4] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, (3):209–223, 1997. 9, 25

[5] David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors. *Graph Partitioning and Graph Clustering, 10th DIMACS Implementation Challenge Workshop, Georgia Institute of Technology, Atlanta, GA, USA, February 13-14, 2012. Proceedings*, Contemporary Mathematics, 2013. American Mathematical Society. 9

[6] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 623–632. SIAM, 2002. 8

[7] Vladimir Batagelj and Matjaz Zaversnik. Fast algorithms for determining (generalized) core groups in social networks. *Advanced Data Analysis and Classification*, (2):129–145, 2011. 21, 27, 34

[8] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 16–24. ACM, 2008. 7

[9] Shankar Bhamidi, Guy Bresler, and Allan Sly. Mixing time of exponential random graphs. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 803–812. IEEE Computer Society, 2008. 7

[10] Binh-Minh Bui-Xuan, Ondřej Suchý, Jan A. Telle, and Martin Vatshelle. Feedback vertex set on graphs of low clique-width. *European Journal of Combinatorics*, (3): 666–679, 2013. 36

[11] Laurent Bulteau, Vincent Froese, Konstantin Kutzkov, and Rasmus Pagh. Triangle counting in dynamic graph streams. *Algorithmica*, (1):259–278, 2016. 8

[12] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, (5):2075–2089, 2010. 8

[13] Derek G. Corneil, Yehoshua Perl, and Lorna K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, (4):926–934, 1985. 24, 43

[14] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, (2):125–150, 2000. 35

[15] Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. In *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, pages 290–301. Springer, 2013. 47, 66

[16] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. 8

[17] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, (6):968–992, 2004. 8

[18] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013. 8, 18

[19] Ronald D. Dutton, Robert C. Brigham, and Fernando Gomez. INGRID: A graph invariant manipulator. *Journal of Symbolic Computation*, (2):163–177, 1989. 65

[20] Jean-Pierre Eckmann and Elisha Moses. Curvature of co-links uncovers hidden thematic layers in the world wide web. *Proceedings of the National Academy of Sciences*, (9):5825–5829, 2002. 7

[21] Lars Engebretsen and Jonas Holmerin. Clique is hard to approximate within $n^{1-o(1)}$. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, pages 2–12. Springer, 2000. 9

[22] David Eppstein and Emma S. Spiro. The h-index of a graph and its application to dynamic subgraph statistics. *Journal of Graph Algorithms and Applications*, (2):543–567, 2012. 33

[23] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 2013. 9, 27

[24] Igor Fabrici. Light graphs in families of outerplanar graphs. *Discrete Mathematics*, (7-8):866–872, 2007. 25

[25] Emilio Ferrara. Measurement and analysis of online social networks systems. In *Encyclopedia of Social Network Analysis and Mining*, pages 891–893. 2014. 20, 25

[26] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. 8

[27] Ove Frank and David Strauss. Markov graphs. *Journal of the American Statistical Association*, (395):832–842, 1986. 7

[28] Tobias Friedrich and Anton Krohmer. Parameterized clique on scale-free networks. In *Algorithms and Computation - 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012. Proceedings*, pages 659–668. Springer, 2012. 9

[29] Ioannis Fudos and Christoph M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, (2): 179–216, 1997. 7

[30] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 22

[31] Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 102–113. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015. 8, 9

[32] Michael T. Goodrich and Pawel Pszona. External-memory network analysis algorithms for naturally sparse graphs. In *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, pages 664–676. Springer, 2011. 9

[33] Carsten Grabow, Stefan Grosskinsky, Jürgen Kurths, and Marc Timme. Collective relaxation dynamics of small-world networks. *CoRR*, 2015. 7

[34] Oded Green and David A. Bader. Faster clustering coefficient using vertex covers. In *International Conference on Social Computing, SocialCom 2013, SocialCom/PASSAT/BigData/EconCom/BioMedCom 2013, Washington, DC, USA, 8-14 September, 2013*, pages 321–330. IEEE Computer Society, 2013. 9, 21, 27, 45

[35] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Science & Business Media, 2012. 41

[36] Frank Gurski. A comparison of two approaches for polynomial time algorithms computing basic graph parameters. *CoRR*, 2008. 36

[37] Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, (1):41–59, 2010. 52

[38] Marijn Heule and Stefan Szeider. A SAT approach to clique-width. *ACM Transactions on Computational Logic*, (3):24, 2015. 23

[39] John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *J. ACM*, (4):549–568, 1974. 25

[40] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, (4):413–423, 1978. 9, 18, 19, 30, 35, 41

[41] Bart M. P. Jansen. *The power of data reduction: Kernels for Fundamental Graph Problems.* PhD thesis, Utrecht University, 2013. 65

[42] Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pages 85–103. Springer, 1972. 21, 22, 23

[43] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. The web as a graph: Measurements, models, and methods. In *COCOON*, pages 1–17. Springer, 1999. 20

[44] Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics*, (1-2):161–185, 2012. 7, 8

[45] Matthieu Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science*, (1-3):458–473, 2008. 7, 8, 17, 18, 31

[46] Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *Journal of the ACM*, (1):1–15, 2002. 8

[47] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Science*, (2):219–230, 1980. 23, 24

[48] Dániel Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, (4): 747–768, 2010. 24

[49] George B. Mertzios, André Nichterlein, and Rolf Niedermeier. Fine-grained algorithm design for matching. *CoRR*, 2016. 9

[50] Sandra L. Mitchell. Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Information Processing Letters*, (5):229–232, 1979. 25

[51] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, (2):415–419, 1985. 9, 18

[52] Mark E. J. Newman. The structure and function of complex networks. *SIAM Review*, (2):167–256, 2003. 7

[53] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006. 8

[54] Donald J. Rose, Robert E. Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, (2):266–283, 1976. 23

[55] Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Experimental and Efficient Algorithms, 4th InternationalWorkshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings*, pages 606–609. Springer, 2005. 7, 8

[56] Thomas Schank and Dorothea Wagner. Approximating clustering coefficient and transitivity. *Journal of Graph Algorithms and Applications*, (2):265–275, 2005. 7, 8

[57] Manuel Sorge and Mathias Weller. The graph parameter hierarchy. Unpublished Manuscript, 2016. 10, 20, 25, 26, 65

[58] Robert Endre Tarjan and Anthony E. Trojanowski. Finding a maximum independent set. *SIAM Journal on Computing*, (3):537–546, 1977. 22

[59] Martin Vatshelle. *New width parameters of graphs.* PhD thesis, The University of Bergen, 2012. 23, 36, 65

[60] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and applications.* Cambridge University Press, 1994. 7

[61] Howard T. Welser, Eric Gleave, Danyel Fisher, and Marc A. Smith. Visualizing the signatures of social roles in online discussion groups. *Journal of Social Structure, 8(2)*, 2007. 7

[62] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012. 14

[63] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654. IEEE Computer Society, 2010. 8, 58, 66

[64] Soon-Hyung Yook, Zoltán N. Oltvai, and Albert-László Barabási. Functional and topological characterization of protein interaction networks. *Proteomics*, (4):928–942, 2004. 8

[65] Yang Zhang and Srinivasan Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pages 1049–1060. IEEE Computer Society, 2012. 8