Technical University of Berlin
Electrical Engineering and Computer Science
Institute of Software Engineering and Theoretical Computer Science
Algorithmics and Computational Complexity (AKT)

Mind the Gap When Searching for Relaxed Cliques

Niklas Wünsche

Thesis submitted in fulfillment of the requirements for the degree
"Master of Science" (M. Sc.) in the field of Computer Science

March 2021

Supervisor and first reviewer:	Prof. Dr. Rolf Niedermeier
Second reviewer:	Prof. Dr. Christian Komusiewicz (Philipps-Universität Marburg)
Co-Supervisors:	Tomohiro Koana and Dr. André Nichterlein

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.

Die selbstständige und eigenhändige Ausfertigung versichert an Eides statt

Berlin, den	Datum	Unterschrift

Zusammenfassung

MAXIMUM CLIQUE – das Finden von Cliquen der maximalen Ordnung $\omega \in \mathbb{N}$ in Graphen der Ordnung $n \in \mathbb{N}$, ist eines der grundlegendsten Probleme der Netzwerkanalyse. Einer der derzeit leistungsfähigsten Algorithmen zum Lösen von MAXIMUM CLIQUE auf Echtwelt-Graphen wurde formal niedergeschrieben von Komusiewicz [Dissertation, 2011] und von Walteros und Buchanan implementiert [Operations Research, 2020]. Er baut auf zwei Zutaten auf:

(1) Aufteilung des Eingabegraphen in viele Teilgraphen, die separat gelöst werden können, und

(2) lösen jedes Teilgraphen, das heißt Finden einer Clique maximaler Ordnung.

Dieser Algorithmus ist schnell wenn die so genannte *Cliquen-Kern-Lücke* zwischen der degeneracy d eines Graphen und ω klein ist, da er in $2^{d-\omega} \cdot n^{\mathcal{O}(1)}$ Schritten ausgeführt werden kann.

Cliquen sind oft zu restriktiv, um ungenaue Daten zu analysieren, daher wurden in der Vergangenheit viele „Clique-Relaxierungs“-Konzepte vorgeschlagen. Ziel dieser Arbeit ist es, den MAXIMUM CLIQUE-Algorithmus von oben auf Clique-Relaxierungen zu verallgemeinern. Dabei werden wir drei Cliquen-Relaxierungen näher betrachten (s-defective cliques, s-plexes, und s-clubs).

Unsere Arbeit besteht aus drei Teilen. Zunächst stellen wir fest, dass einige Clique-Relaxierungen, im Gegensatz zu Clique, wahrscheinlich nicht „fixed-parameter tractable“ hinsichtlich der Cliquen-Kern-Lücke sind, indem wir mehrere NP- und $W[1]$-Härteergebnisse für Clique-Relaxierungen auflisten und die Literatur vervollständigen.

Zweitens passen wir Zutat (2) des MAXIMUM CLIQUE-Algorithmus an, indem wir den kombinierten Parameter „größere Lücke“ plus „jeweiligem Relaxierungsparameter“ $(n-\omega)+s$ betrachten. Wir führen mehrere FPT-Resultate für Cliquen-Relaxierungen auf und stellen neue Algorithmen und Problemkerne für s-defective cliques vor.

Drittens passen wir Zutat (1) des MAXIMUM CLIQUE-Algorithmus an, indem wir eine „mittelgroße“ Lücke präsentieren, für die Cliquen-Relaxierungen noch „fixed-parameter tractable“ sind. Das heißt, wir verwenden einen neuen Parameter $\alpha \in \mathbb{N}$, welcher die degeneracy so verallgemeinert, dass $d \leq \alpha \leq n$ gilt, und wir „fixed-parameter tractable“ für den Parameter $(\alpha - \omega) + s$ sind. Dies führt zu einem Ansatz für das Lösen von MAXIMUM CLIQUE-Relaxierungen, basierend auf den beiden Anpassungen des MAXIMUM CLIQUE-Algorithmus. Wir wenden diesen Ansatz auf s-defective cliques, s-plexes, und s-clubs an. Zum Schluss geben wir auch einen Überblick über die Werte von α und der „mittelgroßen“ Lücken in Echtwelt-Graphen.

Abstract

Maximum Clique – finding cliques of maximum-order $\omega \in \mathbb{N}$ in graphs of order $n \in \mathbb{N}$, is one of the most fundamental problems in network analysis. One of the currently best-performing algorithms for solving Maximum Clique on real-world graphs was formally written down by Komusiewicz [PhD Thesis, 2011] and implemented by Walteros and Buchanan [Operations Research, 2020]. It builds upon two ingredients:

(1) Splitting the input graph into many subgraphs which can be solved separately, and

(2) solving each subgraph, that is, finding a maximum-order clique.

This algorithm performs well when the so-called *clique-core gap* between the degeneracy d of a graph and ω is small, as it runs in $2^{d-\omega} \cdot n^{\mathcal{O}(1)}$ time.

Cliques are often too restrictive to analyze noisy data, hence many "clique relaxations" concepts have been proposed. In this work, our goal is to generalize the Maximum Clique algorithm from above to clique relaxations. We will consider three clique relaxations in more detail (s-defective cliques, s-plexes, and s-clubs).

Our work has three parts. First, we observe that several clique relaxations are, in contrast to clique itself, unlikely to be fixed-parameter tractable with respect to the clique-core gap. We do this by listing several NP-, and $W[1]$-hardness results for clique relaxations and completing the literature.

Second, we adapt ingredient (2) of the Maximum Clique algorithm by considering the combined parameter "larger gap" plus "respective relaxation parameter", $(n-\omega)+s$. Here, we list several FPT-results for clique relaxations, and provide new algorithms and problem kernels for s-defective cliques.

Third, we adapt ingredient (1) of the Maximum Clique algorithm by presenting a "middle ground" *relaxed gap* for which clique relaxations are still fixed-parameter tractable. That is, we employ a new parameter $\alpha \in \mathbb{N}$ generalizing degeneracy such that $d \leq \alpha \leq n$, and we have fixed-parameter tractability for the parameter $(\alpha - \omega) + s$. This leads to a framework for solving Maximum Clique relaxations based on the two adaptations of the Maximum Clique algorithm. We apply this framework to s-defective cliques, s-plexes, and s-clubs. In the end, we also provide an overview of the values of α and the relaxed gaps on real-world graphs.

Contents

Chapter 1

Introduction

Finding "cohesive" parts of a graph is of interest in many different research areas, such as biology [SUS07; Yu+06], social network analysis [Kad68], and even when analyzing the stock market [BBP06; Bog+14]. The most straightforward way to define a cohesive subset of vertices $S \subseteq V(G)$ of a graph G is to require that each pair of vertices of S shares an edge in G, hence forming a *clique*. However, the definition of a clique is very restrictive. For example, small errors in the data can already destroy the "perfect" character of a clique in practice [Yu+06]. Hence, several "relaxed clique"-concepts were proposed, which relax the definition of cliques in one way or another. We provide an overview of them in Table 1.1.

Nevertheless, cliques are very well studied and many algorithms for finding cliques were proposed in the past. Hence, for developing algorithms for finding relaxed cliques, it is of interest to study known clique algorithms [Öst02] and try to generalize them, as has been accomplished in the past [Shi13; Tru+13]. The goal of this thesis is to generalize another algorithm for finding cliques to finding relaxed cliques.

In terms of computational complexity, finding a large clique in a graph is a computationally challenging task, as its decision problem Clique is NP-complete [Kar72], and the corresponding maximization problem Maximum Clique is NP-hard. As Clique is a special case of many Clique relaxations, most of these "relaxed" decision and maximization problems are NP-hard as well.

A very basic, but exact approach for finding maximum-order (relaxed) cliques in a graph is the *Russian-Doll-Search (RDS)* algorithm [CP90; VLS96]. In its most general form, the RDS-algorithm can be stated as follows. For a given graph G:

(1) Define some ordering $(v_1, v_2, \ldots, v_n)$ on the vertices of G, and

(2) for each $i \in [n] := \{1, 2, \ldots, n\}$, based on the results for $j < i$, find a maximum-order (relaxed) clique of the induced subgraph $G[\{v_1, v_2, \ldots, v_i\}]$.

It has been shown that variants of the RDS-algorithm are fast in practice for solving Maximum Clique [Buc+14; Öst02; RGG15; VLS96; WB20], as well as Maximum Clique relaxations [Che+21; GIP18; HKN15a; Shi13; Tru+13]. This is because these algorithms use various (heuristic) data reduction rules, as well as lower bounds [Sto+20] and upper bounds [LW70; Shi13] on the order of a maximum (relaxed) clique, hence they often do not have to consider all $n := |V(G)|$ subgraphs of the RDS-algorithm.

Table 1.1: An overview of clique relaxations, categorized by their relaxation parameter [Gsc+20; Kom16]. Let G be a graph and $S \subseteq V(G)$ induce a respective clique (relaxation). Furthermore, let $\gamma \in [0, 1]$ and, based on the parameter, let $s \in \mathbb{N}$ or $s \in \mathbb{N}_+$. We shorten the names γ-complete graph, s-defective clique, and highly connected subgraph to γ-CG, s-DC, and HCS, respectively. We emphasize the names of the clique relaxations which we will study in this work.

Parameter	Definition	Clique value	Clique relaxation, Reference(s), Value
Minimum Degree $\delta(G)$	$\min_{v\in V(G)} \deg_G(v)$	$\delta(G[S]) = \lvert S\rvert - 1$	s-core [BH03], $\delta(G[S]) \geq s$ *s-plex* [SF78], $\delta(G[S]) \geq \lvert S\rvert - s$ γ-CG [JP09; MIH99], $\delta(G[S]) \geq \gamma \cdot (\lvert S\rvert - 1)$
Average Degree $a(G)$	$\frac{\sum_{v\in V(G)} \deg_G(v)}{\lvert V(G)\rvert}$	$a(G[S]) = \lvert S\rvert - 1$	average s-plex [Guo+11], $a(G[S]) \geq \lvert S\rvert - s$
Distance $\operatorname{dist}_G(u, v)$	length of shortest u-v-path	$\forall u, v \in S :$ $\operatorname{dist}_G(u, v) = 1$	s-clique [Luc50], $\operatorname{dist}_G(u, v) \leq s \forall u, v \in S$ *s-club* [Mok79], $\operatorname{dist}_{G[S]}(u, v) \leq s \forall u, v \in S$
Number $\lvert E(G)\rvert$ of Edges	-	$\lvert E(G[S])\rvert = \binom{\lvert S\rvert}{2}$	s-dense subgraph [KP93; RS08], $\lvert E(G)\rvert \geq s$ *s-DC* [Guo+11; Yu+06], $\lvert E(G)\rvert \geq \binom{\lvert S\rvert}{2} - s$ γ-quasi clique [ARS02], $\lvert E(G)\rvert \geq \gamma \cdot \binom{\lvert S\rvert}{2}$
Vertex Connectivity $\kappa(G)$	$\min_{\substack{V' \subseteq V(G): \\ G - V' \text{ is disconnected}}} \lvert V'\rvert$	$\kappa(G[S]) = \lvert S\rvert - 1$	s-block [Mad78], $\kappa(G[S]) \geq s$ s-bundle [PYB13], $\kappa(G[S]) \geq \lvert S\rvert - s$ γ-RVCS [Ver+14], $\kappa(G[S]) \geq \gamma \cdot (\lvert S\rvert - 1)$
Edge Connectivity $\lambda(G)$	$\min_{\substack{E' \subseteq E(G): \\ G - E' \text{ is disconnected}}} \lvert E'\rvert$	$\lambda(G[S]) = \lvert S\rvert - 1$	γ-RECS [Kom16], $\lambda(G[S]) \geq \gamma \cdot (\lvert S\rvert - 1)$ HCS [HS00], $\lambda(G[S]) \geq \lvert S\rvert /2$
Number $\nu(G)$ of Common Neighbors	$\min_{\{u,v\}\in E(G)} \lvert N_G(u) \cap N_G(v)\rvert$	$\nu(G[S]) = \lvert S\rvert - 2$	s-community [Coh08; VBB15], $\nu(G[S]) \geq s$

We observe that the running time of the RDS-algorithm heavily relies on the ordering of the vertices in step (1) and the algorithm used in step (2). It has been shown that in practice, a *degeneracy ordering* [LW70] for the vertices works well [CP90; HKN15a; Tru+13; WB20]. A degeneracy ordering $(v_1, v_2, \dots, v_n)$ of the vertices of G guarantees that v_i has the minimum degree δ in the induced subgraph $G[v_i, v_{i+1}, \dots, v_n]$. Furthermore, we say that $d(G) := \max_{i\in[n]} |\delta(G[v_i, v_{i+1}, \dots, v_n])|$ is the *degeneracy* of G.

Most variants of the RDS-algorithm lack a (non-trivial) theoretical analysis of their worst-case running time. One of the few RDS-based algorithms for Maximum Clique for which a non-trivial upper bound on the running time is known is the algorithm of Komusiewicz [Kom11, Proposition 5.4]. It has the following two ingredients:

(1) Compute a degeneracy ordering $(v_1, v_2, \dots, v_n)$ for the input graph G, and

(2) for each $i \in [n]$, find a maximum-order clique of the induced subgraph of G on $N[v_i] \cap \{v_i, v_{i+1}, \dots, v_n\}$ by branching over two (non-adjacent) vertices which cannot be part of the same clique, until the resulting instance is a clique.

As, to the best of our knowledge, this folklore algorithm was first formally studied by Komusiewicz in theory and Walteros and Buchanan [WB20] first studied it in practice, we will call it the *KWB-algorithm*.

Note that although we do not necessarily consider G as a whole at some point of the KWB-algorithm, the algorithm is still correct. This is because in a clique S all vertices are adjacent, hence we will find S when we consider the “leftmost” vertex of S with respect to the degeneracy ordering.

Komusiewicz [Kom11, Proposition 5.4] implicitly showed that this algorithm runs in $\mathcal{O}^*(2^{d+1-\omega})$ time, where ω is the maximum-order of a clique in G. Note that we use the $\mathcal{O}^*$-notation to omit polynomial factors in the running time f of an exponential-time algorithm. As $d+1$ is by the definition of the degeneracy an upper bound on ω, Walteros and Buchanan [WB20] called $g := d+1-\omega$ the *clique-core gap* of G. The name comes from the d-core [BH03] of a graph G, which is non-empty if and only if G has degeneracy at least d [WB20]. Note that this algorithm performs well in graphs with small clique-core gap, which is often the case in real-world graphs [WB20].

Focus of the thesis. The goal of this thesis is to generalize the KWB-algorithm. That is, we will provide a framework for several clique relaxations Π which generalizes both ingredients of the KWB-algorithm in such a way that we can find a maximum-order vertex set satisfying Π, and we still provide an upper bound on the running time with respect to some new “relaxed gap”.

We want to show that the ideas presented in this work can be applied to different clique relaxations, so we will study s-defective cliques with $s \in \mathbb{N}$, s-plexes, and s-clubs with $s \in \mathbb{N}_+$ in more detail. A vertex set $S \subseteq V(G)$ is an:

- s-defective clique when $G[S]$ contains at most s non-edges;
- s-plex when $G[S]$ has minimum degree at least $|S| - s$; and
- s-club when $G[S]$ has diameter at most s, where the diameter of $G[S]$ is the maximum distance between two vertices in $G[S]$.

We have chosen these three clique relaxations as on the one hand they all relax a different property of the standard clique (see Table 1.1) and hence represent a variety of different clique relaxations, and are on the other hand also strongly related to each other (an s-defective clique is an $(s+1)$-plex [Shi13; Tru+13]; a connected s-plex is an s-club [PYB13]), which will help us to find common patterns between these three clique relaxations. See Figure 1.1 for examples of these clique relaxations.

These three clique relaxations are also well-motivated on their own. The most obvious relaxation of cliques may be to allow to miss some edges. Such an s-defective clique can be used when the input data is noisy. This happens for example when analyzing protein interactions inside a cell, where false negatives (an edge is missing in the graph although it should be present) occur much more often than false positives (an edge is present in the graph although it should be missing) [Yu+06]. However, it could happen that one vertex is part of all missing edges, and thus it has a relatively low degree in the s-defective clique. When looking for a “cohesive” subnetwork in a social network, one might want to forbid that low-degree vertices are part of the subnetwork, but instead “distribute” the errors by allowing that each member of the subnetwork knows “most” of the other members in it. In this case, it is better to “relax” the minimum degree of the subnetwork, thus searching for s-plexes. For example, Wasserman and Faust [WF94,

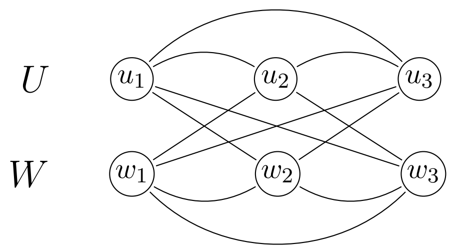


Figure 1.1: A graph G on six vertices which only misses the edges $\{u_i, w_i\}$ for $i \in \{1, 2, 3\}$. Note that U and W induce maximum-order cliques in G. All sets $U \cup \{w_i\}$ and $W \cup \{u_i\}$ induce maximum-order 1-defective cliques. Furthermore, all sets $U \cup \{w_i, w_j\}$ and $W \cup \{u_i, u_j\}$ induce maximum-order 2-defective cliques with $1 \leq i < j \leq 3$. Finally, G is a maximum-order 3-defective clique, 2-plex, and 2-club.

Subsection 7.4.1] used 2-plexes to study real-world business and marriage networks. Finally, one might even allow members of such a subnetwork to not know many other members of the subnetwork directly, as long as they have a common "friend" (or friend of a friend) with every other member inside the subnetwork, thus searching for s-clubs. Milgram [Mil67] claimed that, on average, each citizen of the United States is connected through a "path" of five friends to any other citizen. In other words, the social network of the US contains very large 6-clubs. This phenomenon is known as the *Small-World Problem* [Mil67] or the *Six Degrees of Separation*-idea [EL05].

Note that s-defective cliques as well s-plexes can be disconnected. For example, two isolated vertices induce a 1-defective clique as well as a 2-plex. As Gschwind et al. [Gsc+20] mentioned, it seems like a rather natural "restriction" to exclude disconnected solutions when we search for "cohesive" subgraphs. Hence, we will study these two clique relaxations as well.

Note that we denote the decision (respectively, maximization) problem of finding an s-defective clique/s-plex/s-club of order at least $k \in \mathbb{N}$ (respectively, of maximum order) with s-Defective Clique/s-Plex/s-Club (respectively, Maximum s-Defective Clique/Maximum s-Plex/Maximum s-Club), respectively. When we consider connected s-defective cliques or connected s-plexes, then we will extend the respective problem name with the word Connected.

1.1 Related Work

Finding maximum-order clique relaxations is well-motivated, hence several frameworks for groups of clique relaxations as well as algorithms for special clique relaxations have been proposed in the past. Because we focus in this work on (connected) s-defective cliques, (connected) s-plexes, and s-clubs, we next present related work only for these clique relaxations. We refer to Komusiewicz [Kom16], Singh and Pandey [SP15], Walteros and Buchanan [WB20], and Wu and Hao [WH15] for further information on finding cliques in graphs. See Gschwind et al. [Gsc+17; Gsc+20], Komusiewicz [Kom16], and Pattillo, Youssef, and Butenko [PYB13] for an overview on other clique relaxations.

It is easy to see that s-Club and (Connected) s-Plex are NP-hard even if $s = 1$, and (Connected) s-Defective Clique is NP-hard even if $s = 0$, as these special cases are the well-known Clique problem, which is NP-hard [Kar72].

Note that we say that a problem is *fixed-parameter tractable (in FPT)* with respect to some parameter k when it can be solved in $f(k) \cdot n^{\mathcal{O}(1)}$ time for some computable function f. We note that if a problem parameterized by k is $W[1]$*- or* $W[2]$*-hard*, then it is unlikely to be fixed-parameter tractable under standard parameterized complexity assumptions.

In the following, the parameter $k \in \mathbb{N}$ refers to the solution size of the respective Clique relaxation.

As clique relaxations are well-studied, there is a lot of work from the literature which is related to this thesis. To make it easier for the reader, we provide an overview on the results from the literature which are the most important for this thesis in Table 1.2 on page 19.

(Connected) s-defective cliques. Yu et al. [Yu+06] first introduced s-defective cliques in the context of finding errors in the data of protein interactions inside a cell. To the best of our knowledge, s-defective cliques were first formally defined by Guo et al. [Guo+11]. The s-Defective Clique problem can be decided by a simple brute-force algorithm in $\mathcal{O}^*(2^n)$ time, and Chen et al. [Che+21] provided an algorithm which performs slightly better in the worst case. By a general argument of Lewis and Yannakakis [LY80, Theorem 4], it can be shown that s-Defective Clique is NP-hard for all $s \in \mathbb{N}$. In terms of parameterized complexity, Raman and Saurabh [RS08, Theorem 20] showed implicitly that s-Defective Clique is $W[1]$-hard with respect to the parameter k. On the positive side, Koana, Komusiewicz, and Sommer [KKS20, Theorem 3.7] showed that s-Defective Clique is decidable in $2^{\mathcal{O}(d\sqrt{s})} \cdot n^{\mathcal{O}(\sqrt{s})}$ time, where d is the degeneracy. Moreover, one can easily adapt a result of Komusiewicz [Kom16, Proposition 3] to show that s-Defective Clique is decidable in $2^d \cdot n^{\mathcal{O}(s)}$ time.

Many results for s-Defective Clique with respect to the parameter $n - k$ follow from results of its dual problem, Partial Vertex Cover [BB98; GNW07]. It is known that s-Defective Clique is $W[1]$-complete with respect to $n - k$ due to Guo, Niedermeier, and Wernicke [GNW07, Theorem 14] and Cai [Cai08, Theorem 2.5]. In contrast, s-Defective Clique can be decided in $\mathcal{O}^*(3^{(n-k)+s})$ time by a branching algorithm of Raman and Saurabh [RS08, Theorem 10].

We briefly mention that s-defective cliques have also been considered in weighted graphs [GIP18; Gsc+20; Tru+13] and on restricted graph classes [AFS11; Cas+14; RS08], as well as for many other parameterizations [BGP13; Blä03; KKS20; RS08]. It has also been studied in terms of enumeration algorithms [KKS20], cluster editing variants [Guo+11; SGB14], and graph partitioning and covering problems [Gsc+17; Gsc+20].

For solving Maximum s-Defective Clique in practice for $s \in [1, 4]$, some exact combinatorial branching algorithms [Che+21; GIP18; Shi13; Tru+13] as well as ideas based on integer programming [SS06; Sto+20] have been proposed and implemented. For a comparison on the quality of a combinatorial solver and an integer-programming–based solver, we refer to Stozhkov et al. [Sto+20, Chapter 4].

Connected s-defective cliques on their own are not as well studied as s-defective cliques. One can transfer the hardness results from s-Defective Clique we presented before to Connected s-Defective Clique by adding a new universal vertex which is connected to all other vertices and increasing the solution size k by one. Furthermore, one can adapt the algorithm of Raman and Saurabh [RS08, Theorem 10] to decide Connected s-Defective Clique in $\mathcal{O}^*(3^{(n-k)+s})$ time as well. Moreover, one can easily adapt a result of Komusiewicz [Kom16, Proposition 3] to show that Connected s-Defective Clique is decidable in $2^d \cdot n^{\mathcal{O}(s)}$ time as well. To the best of our knowledge, a result which holds for Connected s-Defective Clique and not for s-Defective Clique is that Connected s-Defective Clique is decidable in $\Delta^{\mathcal{O}(\Delta)} \cdot n^{\mathcal{O}(1)}$ time by applying a framework of Komusiewicz [Kom16, Proposition 1], where Δ is the maximum degree. Moreover, Gschwind et al. [Gsc+20] provided an exact branching algorithm for Maximum Connected s-Defective Clique. However, to the best of our knowledge, no solver for Maximum Connected s-Plex has been implemented yet. One reason why this might be the case is that an s-defective clique of order at least $s+2$ is necessarily connected for all $s \in \mathbb{N}$, as we know due to Pattillo, Youssef, and Butenko [PYB13, Proposition 8.d] (observed by Gschwind et al. [Gsc+20, Table 1]). Hence, we can observe that all maximum-order s-defective cliques we consider in Tables A.5 to A.8 are necessarily connected.

(Connected) s-plexes. Seidman and Foster [SF78] first introduced s-plexes in the context of social network analysis. The s-Plex problem can be decided by a simple brute-force algorithm in $\mathcal{O}^*(2^n)$ time, and Xiao et al. [Xia+17] provided an algorithm which performs slightly better in the worst case. By a general argument of Lewis and Yannakakis [LY80, Theorem 4], it can be shown that s-Plex is NP-hard for all $s \in \mathbb{N}_+$. Kosub [Kos04, Theorem 6.2.3] and Balasundaram, Butenko, and Hicks [BBH11, Theorem 2] gave a direct reduction from the NP-hard Clique to s-Plex for all s. In terms of parameterized complexity, Komusiewicz et al. [Kom+09, Theorem 5] showed that s-Plex is $W[1]$-hard with respect to the combined parameter $k+s$. Furthermore, Koana, Komusiewicz, and Sommer [KKS20, Theorem 3.3] showed that s-Plex is $W[1]$-hard with respect to the combined parameter $k+s+d$, where d is the degeneracy of the graph. On the positive side, Komusiewicz [Kom16, Proposition 2] showed that s-Plex is decidable in $\Delta^{\mathcal{O}(\Delta+s)}$ time by applying a framework of Komusiewicz and Sorge [KS15, Theorem 4]. Moreover, Komusiewicz [Kom16, Proposition 3] showed that s-Plex is decidable in $2^d \cdot n^{\mathcal{O}(s)}$ time.

Many results for s-Plex with respect to the parameter $n-k$ follow from results of its dual problem, Bounded-Degree Vertex Deletion. It is known that s-Plex is $W[2]$-complete with respect to $n-k$ due to Fellows et al. [Fel+11, Theorem 2]. In contrast, s-Plex can be solved in $\mathcal{O}^*(((n-k)+s)^{(n-k)+3})$ time due to Nishimura, Ragde, and Thilikos [NRT05, Theorem 5], as well as in $\mathcal{O}^*((s+1)^{n-k})$ time by a branching algorithm of Komusiewicz et al. [Kom+09, Theorem 6], which was slightly improved in the polynomial part of its running time by Moser, Niedermeier, and Sorge [MNS12, Theorem 2]. Fellows et al. [Fel+11, Corollary 1] also showed that for constant s and any $\varepsilon > 0$, s-Plex admits a problem kernel with $\mathcal{O}((n-k)^{1+\varepsilon})$ vertices which is computable in $\mathcal{O}(n^4 \cdot m)$ time. In the special cases of 1-Plex (which is Clique) and

2-Plex, they showed that their problem kernel only contains a linear number of vertices with respect to $n - k$. Furthermore, for constant s, Moser, Niedermeier, and Sorge [MNS12, Theorem 1] provided a problem kernel with $\mathcal{O}((n-k)^2)$ vertices which is computable in quadratic time.

We briefly mention that s-plexes have been considered in weighted graphs [GIP18; Shi13; Tru+13] and temporal graphs [Ben+19], as well as in terms of enumeration algorithms [Con+17; KKS20; Kom+09], cluster editing variants [BMN12; Guo+11; SGB14], and graph partitioning and covering problems [Gsc+17; Gsc+20].

For solving Maximum s-Plex in practice for $s \in [2, 5]$, many exact combinatorial branching algorithms [Che+20; Con+17; GIP18; HKL19; MH12; MNS12; Pul20; Shi13; Tru+13; Xia+17; ZH17] as well as ideas based on integer programming [BBH11; Gsc+17; Sto+20; ZH17] have been proposed and implemented. For a comparison on the quality of a combinatorial solver and an integer-programming–based solver, we refer to Stozhkov et al. [Sto+20, Chapter 4].

Connected s-plexes on their own are not as well studied as s-plexes. One can transfer the hardness results from s-Plex we presented before to Connected s-Plex by adding a new universal vertex which is connected to all other vertices and increasing the solution size k by one. Furthermore, one can adapt the algorithm of Komusiewicz et al. [Kom+09, Theorem 6] to decide Connected s-Plex in $\mathcal{O}^*((s+1)^{n-k})$ time as well. Moreover, one can easily adapt a result of Komusiewicz [Kom16, Proposition 3] to show that Connected s-Plex is decidable in $2^d \cdot n^{\mathcal{O}(s)}$ time as well. To the best of our knowledge, a result which holds for Connected s-Plex and not for s-Plex is that Connected s-Plex is decidable in $\Delta^{\mathcal{O}(\Delta)} \cdot n^{\mathcal{O}(1)}$ time [Kom16, Section 2.1, Proposition 1]. Moreover, Gschwind et al. [Gsc+20] provided an exact branching algorithm for Maximum Connected s-Plex. However, to the best of our knowledge, no solver for Maximum Connected s-Plex has been implemented yet. One reason why this might be the case is that an s-plex of order at least $2s - 1$ is necessarily connected for all $s \in \mathbb{N}_+$, as we know due to Seidman and Foster [SF78]. Nevertheless, there are real-world graphs which do not contain such "large" s-plexes, and which could therefore only contain maximum-order s-plexes which are disconnected. For example, this is the case for the graphs `belgium.osm` and `ecology1` in Table A.8.

s-clubs. First, we note that for s-clubs there is a difference between finding an s-club of order *exactly* k and *at most* k. As we will study the corresponding maximization problems of clique relaxations in this thesis, we will only consider the decision problem of finding an s-club of order *at least* k, and we refer to the survey of Komusiewicz [Kom16] for a further discussion on finding s-clubs of order *exactly* k.

Mokken [Mok79] first introduced s-clubs in the context of social network analysis. To the best of our knowledge, Bourjolly, Laporte, and Pesant [BLP02] were the first to study the computational complexity of s-Club. They showed that s-Club is NP-complete for all $s \in \mathbb{N}_+$, and provided a branching algorithm. It was later shown that this algorithm has a worst-case running time of $\mathcal{O}^*(2^{n-k})$ [Sch+12] as well as $\mathcal{O}^*(1.62^n)$ [Cha+13]. In terms of parameterized complexity, Schäfer et al. [Sch+12] studied s-Club with respect to the solution size k in more detail. Hartung, Komusiewicz, and Nichterlein [HKN15a] provided several problem kernels, algorithms, and hardness results for s-Club with respect to some structural parameters. Furthermore, Hartung, Komusiewicz, and Nichterlein showed that the branching algorithm of Bourjolly, Laporte, and Pesant [BLP02] and Schäfer et al. [Sch+12] cannot be significantly improved unless the *Strong Exponential Time Hypothesis* [IPZ01] breaks, which would result in a major breakthrough in parameterized complexity theory. For the special case of 2-Club, Hartung et al. [Har+15] and Hartung, Komusiewicz, and Nichterlein [HKN15a] provided even more algorithms and hardness results with respect to some structural parameters. We briefly mention that s-clubs have also been considered in terms of cluster editing variants [Fig+21; LZZ12; MPS20], as well as graph partitioning and covering problems [Gsc+17; Gsc+20].

Chang et al. [Cha+13], Hartung, Komusiewicz, and Nichterlein [HKN15a], and Komusiewicz et al. [Kom+19] provided implementations based on the (combinatorial) branching algorithm of Bourjolly, Laporte, and Pesant [BLP02] for 2-Club with additional heuristics, which all perform well on real-world graphs. A comparison of the three algorithms was given by Komusiewicz et al. [Kom+19, Table 3]. For $s \in [2, 5]$, further exact approaches, mostly based on (mixed) integer programming, have been studied to solve Maximum s-Club on small or random graph instances [AC12; BBT05; BLP02; CA11; Gsc+20; PB12; SB20; VB12; YPB17]. For 2-clubs, Hartung, Komusiewicz, and Nichterlein [HKN15a, Table 6] showed that their branching algorithm performs better on real-world graphs than a respective integer programming approach.

Degeneracy variants. To generalize the KWB-algorithm, we will generalize the degeneracy in Chapter 4. To the best of our knowledge, the degeneracy of a graph was first defined as the *coloring number* of a graph (not to be confused with the *chromatic number* of a graph) by Erdős and Hajnal [EH66]. It was re-introduced by Lick and White [LW70, Chapter 2], and Matula and Beck [MB83] were the first to provide a linear-time algorithm for computing the degeneracy of a graph. Bader and Hogue [BH03] re-introduced the degeneracy again as the core-number of a graph.

Many generalizations of the degeneracy have been proposed in the past [Ben+19; Buc+14; KWB05; Zak13]. One generalization which will be of interest for us is the *weak x-degeneracy ordering* of a graph G with $x \in \mathbb{N}$ [Pic15; Tru+13]. A weak x-degeneracy ordering $(v_1, v_2, \dots, v_n)$ of the vertices of G guarantees that v_i has the minimum x-degree δ_x in $G[v_i, v_{i+1}, \dots, v_n]$, where the x-degree of a vertex v_i is the number of vertices with distance at most x to v_i in $G[v_i, v_{i+1}, \dots, v_n]$, except v_i itself. Furthermore, $\alpha_x(G) = \max_{i \in [n]} |\delta_x(G[v_i, v_{i+1}, \dots, v_n])|$ is the *weak x-degeneracy* of G. Note that the degeneracy is equal to the weak 1-degeneracy of a graph. As in a relaxed clique not all vertices are necessarily adjacent to each other (otherwise it would form a standard clique), this generalization of the standard degeneracy will help us to generalize ingredient (1) of the KWB-algorithm.

There is also another natural generalization of the standard degeneracy which we call the *strong x-degeneracy* of a graph, see Definition 5.3 for a formal definition. However, we will not study it in more detail because we show in Proposition 5.5 that the weak x-degeneracy of a graph is at most its strong x-degeneracy, but the converse is not true.

We mention that Picker [Pic15, Section 4.2] already briefly introduced the notation of the weak 2-degeneracy of a graph in the context of a variant of 2-Club. However, he assumed that in practice it might take too much time to compute the weak 2-degeneracy ordering of a graph. For this reason, he did not study this new parameter in more detail. Furthermore, Trukhanov et al. [Tru+13, Subsection 3.3.3] also briefly introduced the notion of a weak 2-degeneracy ordering in the context of the s-Plex problem. However, Trukhanov et al. [Tru+13, Chapter 4] mention that for their RDS-algorithm, a (faster to compute) standard degeneracy ordering performed better in practice. Nevertheless, trying to generalize the result that Clique is fixed-parameter tractable with respect to the clique-core gap (due to the KWB-algorithm) to Clique relaxations will motivate us to study the weak x-degeneracy of a graph in more detail.

1.2 Our Contributions

The results of this work can be divided into three parts. First, we show that the Clique relaxation s-Defective Clique with $s \in \mathbb{N}$ is $W[1]$-hard with respect to the clique-core gap $g := d+1-\omega$ (Corollary 3.6 on page 32), We also show this parameterized problem is contained in $W[1]$ (Theorem 3.10 on page 34).

To circumvent this hardness results, second we study s-Defective Clique with respect to the combined parameter $(n-k)+s$ in more detail by using the duality of the problems s-Defective Clique and Partial Vertex Cover and applying techniques already known in the context of Vertex Cover. Note that for Partial Vertex Cover, one asks whether a graph G contains a vertex cover of size at most $k \in \mathbb{N}$ which does not cover at most $s \in \mathbb{N}$ edges. Furthermore, note that a graph G contains an s-defective clique of order $n-k$ if and only if the complement graph $\overline{G}$ contains an s-partial vertex cover of size k. We will provide two problem kernels for s-Defective Clique with respect to the parameter $(n-k)+s$, that is, one can reduce the size of an s-Defective Clique instance in linear time (quadratic time, respectively) such that the resulting s-Defective Clique instance contains $\mathcal{O}((n-k)^2)$ vertices ($\mathcal{O}(n-k)$ vertices, respectively) for constant s (Theorems 4.8 and 4.9 on pages 42 and 43). The linear-vertex problem kernel is one of the highlights of this thesis, as it fills the hole between the Clique problem (for which a linear-vertex problem kernel exists with respect to $n-k$ [CKJ01; NT75]), and the strongly related s-Plex problem (for which a quasi-linear–vertex problem kernel exists with respect to $n-k$ for constant s [Fel+11]). Along the way, we also provide a new branching algorithm for s-Defective Clique (Proposition 4.3 on page 40).

To look at a smaller gap than $n-k$ again, third we provide a generalization of the KWB-algorithm (for solving Maximum Clique) to Maximum Clique relaxations. Along the way, we will study the weak x-degeneracy α_x of a graph for $x \in \mathbb{N}$. We show that this new parameter family is strongly related to the maximum degree of a graph (Proposition 5.5 on page 62). Furthermore, we show that for arbitrary x it holds that α_x is computable in $\mathcal{O}(\alpha_x nm)$ time (Lemma 5.6 on page 63), but for all $\varepsilon > 0$ cannot be computed in $\mathcal{O}(n^{2-\varepsilon})$ time unless the Strong Exponential Time Hypothesis [IPZ01] breaks (Theorem 5.11 on page 65), which would lead to a major breakthrough in parameterized complexity theory. Finally, as a highlight of this thesis, we will provide the *gap-framework* for solving Maximum Clique relaxations (Theorem 5.19 on page 72). This gap-framework, which is a generalization of the KWB-algorithm, can be applied to any clique relaxation Π (and its maximization problem) when the following hold:

(1) There exists an upper-bound x on the diameter of every graph satisfying Π, and

(2) one can verify whether a graph satisfies Π, or find a set of at most $c \in \mathbb{N}$ vertices which cannot be part of the same subgraph satisfying Π, in polynomial time.

If these conditions holds, then one can apply our gap-framework to construct an algorithm for solving the respective Maximum Clique relaxation in $\mathcal{O}^*(c^{(\alpha_x+1)-\omega_\Pi})$ time, where ω_Π is the maximum order of a respective relaxed clique in the input graph.

We apply this gap-framework to Maximum s-Club, Maximum Connected s-Plex, and Maximum Connected s-Defective Clique (Theorems 6.1 to 6.3 on page 74). Furthermore, we provide adaptations of our framework which might perform better in practice when certain additional criteria are met for Maximum (Connected) s-Defective Clique and Maximum (Connected) s-Plex (Sections 6.1.2 and 6.2.2 on pages 77 and 82). For an overview of our results, see Table 1.2.

Structure of the work. In Chapter 2, we will provide the relevant definitions from graph theory, parameterized complexity theory, and the problems we study in this work. In Chapter 3, we will show that, in contrast to the standard Clique problem, it is unlikely that under standard parameterized complexity assumption there exist FPT-algorithms for the Clique relaxations s-Defective Clique, s-Plex, and s-Club with respect to the clique-core gap $g := d + 1 - \omega$. In Chapter 4, we will study the larger "gap" $n - k$ and provide results from the literature that show that all three Clique relaxations are fixed-parameter tractable with respect to the combined parameter "large gap" plus "relaxation parameter" $(n - k) + s$. We also provide three new FPT-results for s-Defective Clique with respect to $(n - k) + s$. In Chapter 5, we will consider a smaller gap by introducing a generalization of the standard degeneracy, and provide a gap-framework (based on the KWB-algorithm) which can be used to solve several Maximum Clique relaxations. In Chapter 6, we will apply our gap-framework and simple modifications of it to Maximum (Connected) s-Defective Clique, Maximum (Connected) s-Plex, and Maximum s-Club. Furthermore, we provide the respective "gap" values for many real-world graphs in Appendix A.

Table 1.2: Overview of previously known results and our most important results for deciding/solving (Maximum) Clique (relaxations) and computing the weak x-degeneracy of a graph. We use the following notation: n–number of vertices, m–number of edges, k–solution size, s–respective relaxation parameter, $x \in \mathbb{N}$, α_x-weak x-degeneracy, ω–maximum-order of a respective (relaxed) clique in input graph, $g_x := \alpha_x + 1 - \omega$, $g_* := g_{\left\lfloor\sqrt{2s+\frac{1}{4}}+\frac{1}{2}\right\rfloor}$, $\varepsilon > 0$, Δ–maximum degree. We note that results or open questions which are centered between two problems apply to both of them. Moreover, all algorithms with respect to some "gap" can also be used to solve the corresponding maximization problem. Furthermore, *PK* stands for problem kernel. For a parameterized problem, *FPT?* denotes that it is open whether this parameterized problem is in FPT.

Parameter	Result	Reference
	(Maximum) Clique	
$n-k$	$\mathcal{O}(2^{n-k}\cdot(n+m)+n^2)$	[Meh84]
	$2\cdot(n-k)$-vertex PK	[CKJ01; NT75]
g_1	$\mathcal{O}(2^{g_1}\cdot\alpha_1^5\sqrt{\alpha_1}n+\omega n)$	[Kom11]
	$\mathcal{O}((1.28^{g_1}+\alpha_1^2)\cdot(n-\alpha_1))$	[WB20]
	(Maximum) s-Club	
$n-k$	$\mathcal{O}(2^{n-k}\cdot nm)$	[Sch+12]
g_1+s	$W[1]$-hard	[Har+15]
g_s	$\mathcal{O}(2^{g_s}\cdot n\cdot\alpha_s^3+\alpha_s nm)$	Theorem 6.3
	(Maximum) s-Plex	
	General	Connected
$n-k$	$W[2]$-hard [Fel+11]	
$(n-k)+s$	$\mathcal{O}((s+1)^{n-k}\cdot(n+m))$[Kom+09]	
	$\mathcal{O}((n-k)^2)$-vertex PK for fix s [MNS12]	
	$\mathcal{O}((n-k)^{1+\varepsilon})$-vertex PK for fix s [Fel+11]	
g_1+s	$W[1]$-hard [KKS20]	
g_2+s	FPT?	
	$\mathcal{O}(((s+1)^{g_2}\cdot n\cdot\alpha_2^2+\alpha_2 nm)+n^{2s-1}\cdot s^2)$ Theorem 6.15	
g_s+s	FPT?	$\mathcal{O}((s+1)^{g_s}\cdot n\cdot\alpha_s^2+\alpha_s nm)$ Theorem 6.2
	(Maximum) s-Defective Clique	
	General	Connected
$n-k$	$W[1]$-hard [Cai08; GNW07]	
$(n-k)+s$	$\mathcal{O}(3^{(n-k)+s}\cdot n^2)$[RS08]	
	$\mathcal{O}((2\cdot(s+1))^{n-k}\cdot n^2)$ Proposition 4.3	
	$\mathcal{O}((n-k)^2)$-vertex PK for fix s Theorem 4.8	
	$\mathcal{O}(n-k)$-vertex PK for fix s Theorem 4.9	
g_1+s	$W[1]$-hard Corollary 3.6	
g_2+s	FPT?	
	$\mathcal{O}(((2\cdot(s+1))^{g_2}\cdot n\cdot\alpha_2^2+\alpha_2 nm)+n^{s+2}\cdot s^2)$ Theorem 6.7	
g_*+s	FPT?	$\mathcal{O}((2\cdot(s+1))^{g_*}\cdot n\cdot\alpha_*^2+\alpha_* nm)$ Theorem 6.1
		$\mathcal{O}(3^{g_*+s}\cdot n\cdot\alpha_*^2+\alpha_* nm)$ Theorem 6.11

Weak x-Degeneracy	
Result	Reference
$\Delta\le\alpha_x\le\Delta^{x+1}$ for $x\ge 2$	Proposition 5.5
$\mathcal{O}(\alpha_x nm)$-time computable	Lemma 5.6
$\mathcal{O}(n+m)$-time computable for $x=1$	[MB83]
No $\mathcal{O}(n^{2-\varepsilon})$-time algorithm unless SETH [IPZ01] breaks	Theorem 5.11

Chapter 2

Preliminaries

General preliminaries. We denote with $\mathbb{N}$ the set of all non-negative natural numbers, starting with 0. Furthermore, $\mathbb{N}_+ := \mathbb{N} \setminus \{0\}$. Moreover, $[n]$ is the set $\{1, 2, \dots, n\}$ for any $n \in \mathbb{N}$.

Let $S := \{s_1, s_2, \dots, s_n\}$ be a set with $n \in \mathbb{N}$, and $\preceq := (s_1, s_2, \dots, s_n)$ be a linear ordering of S. For two elements $s, t \in S, s \neq t$, we say that t is *right of* s with respect to $\preceq$ when $s \preceq t$. For $\emptyset \subsetneq S' \subseteq S$, we say that an element $s \in S'$ is the *leftmost element of* S' with respect to $\preceq$ if $s \preceq t$ for all $t \in S'$.

To omit polynomial factors in the running time f of an exponential-time algorithm, we use the notation $\mathcal{O}^*(f) := \mathcal{O}(f \log^c f)$ for a constant $c \in \mathbb{N}$.

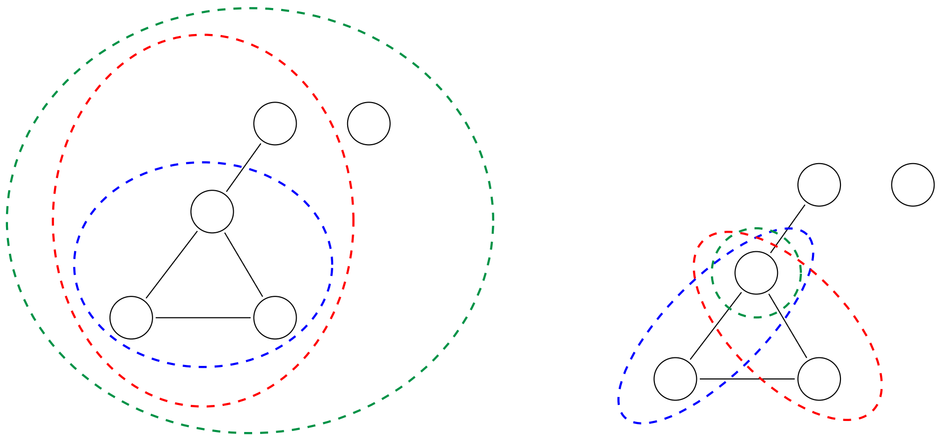
Graph theory. Most of the notation is based upon the textbook of Diestel [Die16]. We denote with G an undirected graph where $V(G)$ denotes the set of vertices and $E(G) \subseteq \{\{v, w\} \mid v, w \in V(G), v \neq w\}$ denotes the set of edges. When not stated otherwise, we only work with undirected graphs in this thesis.

Graph notation. Let G denote an undirected graph. We denote by

$V(G)$	the *vertex set* of G;
$E(G)$	the *edge set* of G; for an edge $e = \{u, v\} \in E(G)$ the two vertices u and v are called *endpoints* of e;
$E(G, A, B)$	the subset of edges of G which have one endpoint in $A \subseteq V(G)$ and one endpoint in $B \subseteq V(G)$, formally $E(G, A, B) := \{\{u, v\} \in E(G) \mid u \in A, v \in B\}$;
n_G	the number $\lvert V(G)\rvert$ of *vertices*;
m_G	the number $\lvert E(G)\rvert$ of *edges*;
$G' \subseteq G$	G' is a *subgraph* of G, formally, G' is a graph with $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$;
$\binom{V(G)}{2}$	the set of all possible edges over $V(G)$, formally, $\binom{V(G)}{2} := \{\{u, v\} \mid u, v \in V(G); u \neq v\}$;

$\overline{G}$	the *complement graph* of G, formally, $V(\overline{G}) := V(G)$, $E(\overline{G}) := \binom{V(G)}{2} \setminus E(G)$;
$G[V']$	the *induced subgraph* of G on the vertex set $V' \subseteq V(G)$, formally, $G[V']$ is the graph with $V(G[V']) := V'$, $E(G[V']) := E(G) \cap \binom{V'}{2}$;
$G - V'$	the *graph obtained by removing the vertices from the vertex set* $V' \subseteq V(G)$ *from* G, formally, $G - V' := G[V(G) \setminus V']$;
$G - v$	the *graph obtained by removing the vertex* $v \in V(G)$ *from* G, formally, $G - v := G[V(G) \setminus \{v\}]$;
$N_G(v)$	the (open) *neighborhood* of v, formally, $N_G(v) := \{u \in V(G) \mid \{u, v\} \in E(G)\}$;
$N_G(V')$	the (open) *neighborhood* of V' for some vertex set $V' \subseteq V(G)$, formally, $N_G(V') := \{u \in V(G) \mid \exists v \in V' : u \in N_G(v)\}$;
$N_G[v]$	the *closed neighborhood* of v, formally, $N_G[v] := N_G(v) \cup \{v\}$;
$\deg_G(v)$	the *degree* of $v \in V(G)$, formally, $\deg_G(v) := \lvert N_G(v)\rvert$; if $\deg_G(v) = 0$, then v is *isolated*;
$\delta(G)$	the *minimum degree* of G, formally, $\delta(G) := \min_{v \in V(G)}\{\deg_G(v)\}$;
$\mathrm{dist}_G(u, v)$	the *distance* between two vertices u and v, formally, $\mathrm{dist}(u, v) :=$ length of a shortest u-v-path;
$\mathrm{diameter}(G)$	the *diameter* of a graph G is the length of a longest shortest path between any two vertices of G;
G^x	the *power graph* on $V(G)$ in which two distinct vertices u and v are connected by an edge if and only if they have distance at most x in G, formally, $V(G^x) := V(G)$, $E(G^x) := \bigcup_{u \in V(G)}\{\{u, v\} \mid 0 < \mathrm{dist}_G(u, v) \leq x\}$ for some $x \in \mathbb{N}_+$;
$N_{x,G}(v)$	the (open) *x-neighborhood* of v for any $x \in \mathbb{N}_+$, formally, $N_{x,G}(v) := \{u \in V \mid u \neq v; \mathrm{dist}_G(u, v) \leq x\}$;
$\deg_{x,G}(v)$	the *x-degree* of v for any $x \in \mathbb{N}_+$, formally, $\deg_{x,G}(v) := \lvert N_{x,G}(v)\rvert$; and
$\delta_x(G)$	the *minimum x-degree* of G, formally, $\delta_x(G) := \min_{v \in V(G)}\{\deg_{x,G}(v)\}$.

If the graph G is clear from the context, then we will omit the subscript or argument G. For example, we write $\deg_x(v)$ instead of $\deg_{x,G}(v)$ and δ instead of $\delta(G)$.



(a) Vertex sets inducing relaxed cliques (b) Vertex sets covering edges

Figure 2.1: Figures 2.1a and 2.1b both show the same graph G with five vertices and two connected components.
Figure 2.1a: The vertex set of the blue subgraph of G induces a maximum-order clique, and hence by definition a maximum-order 0-defective clique, a 1-plex, a 1-club, and a minimum-order 3-dense subgraph in G. The vertex set of the orange subgraph of G induces a maximum-order 2-defective clique, a 3-plex, a 2-club, and a minimum-order 4-dense subgraph in G. The vertex set of the whole, green, graph G induces a maximum-size 6-defective clique and a 5-plex in G.
Figure 2.1b: The vertex sets of the blue and the orange subgraphs of G both form a minimum-size vertex cover and 0-partial vertex cover in G. The singleton vertex set of the green subgraph of G forms a minimum-size 1-partial vertex cover in G. The empty vertex set forms a minimum-size 4-partial vertex cover in G.

Special vertex sets. Let G be a graph. A vertex set $S \subseteq V(G)$ is called a(n)

clique	if $\{u, v\} \in E(G)$ for all $u, v \in S$ with $u \neq v$. Alternatively, $\lvert E(G[S])\rvert = \binom{\lvert S\rvert}{2}$; if $S = V(G)$, then G is the *complete* graph on n vertices, denoted with K_n [GJ79; LP49];
s-defective clique	if for $s \in \mathbb{N}$ it holds that $\lvert E(G[S])\rvert \geq \binom{\lvert S\rvert}{2} - s$ [Guo+11; Yu+06];
s-plex	if for $s \in \mathbb{N}_+$ it holds that $\delta(G[S]) \geq \lvert S\rvert - s$ [SF78];
s-club	if for $s \in \mathbb{N}_+$ it holds that diameter$(G[S]) \leq s$ [Mok79];
ℓ-dense subgraph	if for $\ell \in \mathbb{N}$ it holds that $E(G[S]) \geq \ell$ [FS97; KP93];
vertex cover	if S *covers* all edges of G, formally, $E(G - S) = \emptyset$ [GJ79]; and
s-partial vertex cover	if S covers all but at most s edges of G, formally, $\lvert E(G - S)\rvert \leq s$ [BB98; RS08].

Examples for all of these vertex sets are shown in Figure 2.1.

Special graphs. Let G be graph. We say that G is a

forest	if G contains no cycle;
tree	if G is a connected forest;
path	if $V(G) := \{v_1, v_2, \ldots, v_n\}$, $E(G) := \{\{v_i, v_{i+1}\} \mid i \in [n-1]\}$, where $n \in \mathbb{N}$; we denote with P_n the path on n vertices and say that P_n has *length* $n-1$;
bipartite graph	if $V(G)$ can be partitioned into two vertex sets $V_A(H)$, $V_B(H)$ so that each edge of G has one endpoint in $V_A(G)$ and one endpoint in $V_B(G)$, formally $V(G) = V_A(G) \uplus V_B(G)$, $E(G) = E(G, V_A(G), V_B(G))$. We assume $V_A(G)$ and $V_B(G)$ to be fixed, and often denote a bipartite graph with H;
complete bipartite graph	if G is a bipartite graph, and if all edges between both partitions are present, denoted with $K_{s,t}$ where $s := \lvert V_A(G)\rvert$ and $t := \lvert V_B(G)\rvert$; and
star	if G is a complete bipartite graph such that $s = 1$; we say that the single vertex inside $V_A(G)$ is the *center* of the star, and all t vertices in $V_B(G)$ are the *leaves* of the star.

Graph Parameters. Let G denote an undirected graph. We denote by

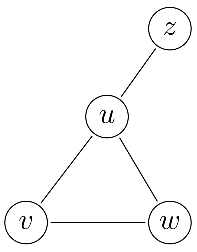
$d(G)$ the *degeneracy* of a graph G, which is the minimum number $d \in \mathbb{N}$ so that every subgraph $G' \subseteq G$ has minimum degree $\delta(G') \leq d$; this definition is equivalent to

$$d(G) := \min_{\preceq \text{ a linear ordering of } V(G)} \{ \max_{v \in V(G)} \{|N_G(v) \cap \{u \in V(G) \setminus \{v\} \mid v \preceq u\}|\}\};$$

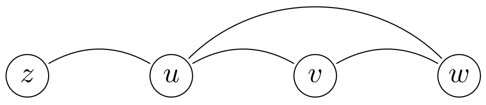
we call a corresponding linear ordering $\preceq_d := (v_1, v_2, \ldots, v_n)$ a *degeneracy ordering* of G; we say that $\mathrm{rN}(v_i, G, \preceq_d) := N_G(v_i) \cap \{v_{i+1}, v_{i+2}, \ldots, v_n\}$ is the (open) *right-neighborhood* of v_i with respect to $\preceq_d$, and $\mathrm{rN}[v_i, G, \preceq_d] := \mathrm{rN}(v_i, G, \preceq_d) \cup \{v_i\}$ the *closed right-neighborhood* of v_i, an example is given in Figure 2.2;

$\omega(G)$ the *maximum order of a clique* in a graph G is the maximum number $\omega \in \mathbb{N}$ such that G contains a clique induced by ω vertices; and

$g(G)$ the *clique-core gap* of a graph G is the (non-negative) difference between the upper bound on the order of a clique $d(G) + 1$ and the actual order $\omega(G)$ of a maximum-order clique in the graph, formally, $g(G) := d(G) + 1 - \omega(G)$. The name comes from the k-core of a graph G, which is non-empty if and only if G has degeneracy at least $k \in \mathbb{N}$ [WB20].



(a) A graph G



(b) G rearranged with respect to a degeneracy ordering of G

Figure 2.2: Figure 2.2a is the illustration of a graph G with four vertices. The graph has degeneracy at least two, because G contains the triangle uvw and in each linear ordering of the vertices, one of the vertices of the triangle has both of the other vertices of the triangle as right-neighbors.
Figure 2.2b illustrates an ordering of the vertices of G. Each vertex has at most two right-neighbors with respect to this ordering, so G has degeneracy two and Figure 2.2b indeed illustrates a degeneracy ordering of G. The right-neighborhoods with respect to the degeneracy ordering are $\mathrm{rN}(z) := \{u\}$, $\mathrm{rN}(u) := \{v, w\}$, $\mathrm{rN}(v) := \{w\}$, and $\mathrm{rN}(w) := \emptyset$.

2.1 Parameterized Complexity

In parameterized complexity theory, we study which parameters make a problem hard and which parameters allow for efficient algorithms. There are many books for parameterized complexity theory [Cyg+15; DF13; FG06; Nie06], and we will build upon Cygan et al. [Cyg+15, Chapters 1-3, 13, Section 9.4].

Let Σ be a finite alphabet and Σ^* be the set of all finite words over Σ. We say that $P \subseteq \Sigma^*$ is a *decision problem*. Furthermore, an *instance* $I \in \Sigma^*$ is a *YES-instance* of P if $I \in P$, and otherwise I is a *NO-instance* of P.

A *parameterized problem* is a subset $\Pi \subseteq \Sigma^* \times \mathbb{N}$. We say that an *instance* $(I, k) \in \Sigma^* \times \mathbb{N}$ is *parameterized* by k. Furthermore, we say that Π is *fixed-parameter tractable (FPT)* if it can be decided in $f(k) \cdot |I|^{\mathcal{O}(1)}$ time whether (I, k) is a YES- or NO-instance of Π for some computable function f only depending on k. The corresponding algorithm is an *FPT-algorithm*. FPT is the class of all parameterized problems which are fixed-parameter tractable. Analogously, XP is the class of all parameterized problems which can be decided in $|I|^{f(k)}$ time for some computable function f.

Showing fixed-parameter intractability. To show that a parameterized problem is (presumably) not in FPT, a hierarchy $\mathrm{FPT} \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[P] \subseteq \mathrm{XP}$ of classes has been established [DF13; FG06]. For us, the most important class (next to FPT) is $W[1]$. A widely believed assumption in parameterized complexity theory is $\mathrm{FPT} \neq W[1]$, hence we can show that a parameterized problem Π is (presumably) not in FPT by doing a *parameterized reduction* from a $W[1]$-hard problem to Π.

A *parameterized reduction* from a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ to a parameterized problem $\Pi' \subseteq \Sigma^* \times \mathbb{N}$ is a function $(\Sigma^* \times \mathbb{N}) \to (\Sigma^* \times \mathbb{N})$ which maps an instance (I, k) of Π to an instance (I', k') of Π' such that: (I, k) is a YES-instance of Π

if and only if (I', k') is a YES-instance of Π', $f(I, k)$ is computable in $g(k) \cdot |I|^{\mathcal{O}(1)}$ time for some computable function g, and $k' \leq h(k)$ for some computable function h.

Another possibility to show that a parameterized problem Π is (presumably) not in FPT is to show that Π is NP-hard even for a constant parameter value. This can be done by doing a standard, polynomial-time reduction from an NP-hard problem to Π. In this case, Π is *para-NP-hard*. Note that a para-NP-hard problem is in FPT if and only if $P = NP$ [FG06, Corollary 2.13].

On the positive side, many techniques to develop FPT-algorithms have been proposed as well [Cyg+15; Fom+18]. Next, we will present the most important techniques we will use in this work.

Problem kernels. Let $\Pi \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that two instances $(I, k), (I', k') \subseteq \Sigma^* \times \mathbb{N}$ are *equivalent* when (I, k) is a YES-instance of Π if and only if (I', k') is a YES-instance of Π. A *data reduction rule* is a polynomial-time computable function $(\Sigma^* \times \mathbb{N}) \rightarrow (\Sigma^* \times \mathbb{N})$ which maps an instance (I, k) of Π to an instance (I', k') of Π. We say that a data reduction rule is *safe* or *correct* if (I, k) and (I', k') are equivalent. We say that a data reduction rule f is *exhaustively applied* to an instance (I, k) of Π if $f(I, k) = (I, k)$. Moreover, if after applying (a polynomial number of) data reduction rules f it holds that $|(I', k')| \leq g(k)$ for some computable function g, then f is a *kernelization* of Π, and we say that Π admits a *problem kernel* of *size* g.

Note that a (decidable) parameterized problem is fixed-parameter tractable if and only if it admits a problem kernel [Cai+97]. We refer to the book of Fomin et al. [Fom+18] for a more in-depth view on constructing problem kernels.

Turing kernels. Problem kernels are a good technique to separate the data reduction rules that lead to one small problem instance from the brute-force algorithm solving this small instance. However, Cygan et al. mentioned that in practice, one does not care whether we solve *one* instance by brute-force at the end of our procedure, or rather many smaller instances along the way, which leads to the notion of Turing kernels.

To the best of our knowledge, Turing kernels were first formally defined by Binkele-Raible et al. [Bin+12], although the idea behind the concept was already known before [Est+05]. Before we can formally define a Turing kernel, first we need to recall the notion of t-oracles according to Binkele-Raible et al. [Bin+12, Defintion 2.3]: A *t-oracle* for a parameterized problem Π is an oracle that takes an instance (I, k) with $|I| \leq t, k \leq t$ as input, and decides whether $(I, k) \in \Pi$ in constant time.

Now we are able to define the notion of a Turing kernel.

Definition 2.1 ([Bin+12, Definition 2.4]). A parameterized problem Π is said to have a *$g(k)$-sized Turing kernel* if there is an algorithm that, given an instance (I, k) together with a $g(k)$-oracle for Π, decides whether $(I, k) \in \Pi$ in $|I, k|^{\mathcal{O}(1)}$ time.

Although not explicitly mentioned by Binkele-Raible et al., we assume that g is some computable function only depending on k.

2.2 Problem Definitions

The following is a list of the relevant problems for this thesis.

Decision problems of clique (relaxations).

CLIQUE [GJ79; LP49]
Input: A graph G and an integer $k \in \mathbb{N}$.
Question: Is there a vertex set $S \subseteq V(G)$ such that $|S| \geq k$ and S is a clique in G?

s-DEFECTIVE CLIQUE [GUO+11; YU+06]
Input: A graph G and integers $k, s \in \mathbb{N}$.
Question: Is there a vertex set $S \subseteq V(G)$ such that $|S| \geq k$ and S is an s-defective clique in G?

s-PLEX [SF78]
Input: A graph G and integers $k \in \mathbb{N}, s \in \mathbb{N}_+$.
Question: Is there a vertex set $S \subseteq V(G)$ such that $|S| \geq k$ and S is an s-plex in G?

s-CLUB [MOK79]
Input: A graph G and integers $k \in \mathbb{N}, s \in \mathbb{N}_+$.
Question: Is there a vertex set $S \subseteq V(G)$ such that $|S| \geq k$ and S is an s-club in G?

DENSE SUBGRAPH [FS97; KP93]
Input: A graph G and integers $k, \ell \in \mathbb{N}$.
Question: Is there a vertex set $S \subseteq V(G)$ with $|S| \leq k$ such that S is an ℓ-dense subgraph in G?

Maximization problems of clique (relaxations).

MAXIMUM CLIQUE [GJ79; LP49]
Input: A graph G.
Question: A maximum-size vertex set $S \subseteq V(G)$ such that S is a clique in G.

MAXIMUM CONNECTED s-DEFECTIVE CLIQUE [GUO+11; YU+06]
Input: A graph G and an integer $s \in \mathbb{N}$.
Question: A maximum-size vertex set $S \subseteq V(G)$ such that $G[S]$ is connected and S is an s-defective clique in G.

MAXIMUM CONNECTED s-PLEX [SF78]
Input: A graph G and an integer $s \in \mathbb{N}_+$.
Question: A maximum-size vertex set $S \subseteq V(G)$ such that $G[S]$ is connected and S is an s-plex in G.

MAXIMUM s-CLUB [MOK79]
Input: A graph G and an integer $s \in \mathbb{N}_+$.
Question: A maximum-size vertex set $S \subseteq V(G)$ such that S is an s-club in G.

Edge-Covering decision problems.

Vertex Cover [GJ79]

Input: An undirected graph G and an integer $\kappa \in \mathbb{N}$.

Question: Is there a vertex set $S \subseteq V(G)$ such that $|S| \leq \kappa$ and S is a vertex cover in G?

Partial Vertex Cover [BB98; RS08]

Input: A graph G and integers $\kappa, s \in \mathbb{N}$.

Question: Is there a vertex set $S \subseteq V(G)$ such that $|S| \leq \kappa$ and S is an s-partial vertex cover in G?

Note that this definition of Partial Vertex Cover is non-standard, as the maximum number s of edges not covered is part of the input, rather than the (standard) minimum number $t := m - s$ of covered edges [BB98; GNW07]. However, our definition improves the readability throughout the thesis, and it is easy to see that both definitions can be easily interchanged.

Chapter 3

The Clique-Core Gap is Too Small

The KWB-algorithm of Komusiewicz [Kom11, Proposition 5.4] (re-introduced and implemented by Walteros and Buchanan [WB20, Theorem 1]) shows that Clique is fixed-parameter tractable with respect to the clique-core gap $g := d + 1 - \omega$, where d is the degeneracy of G, and ω is the maximum-order of a clique in G. Hence, the most obvious path for us to explore when trying to generalize the KWB-algorithm is whether different Clique relaxations are fixed-parameter tractable with respect to g as well.

In this chapter, we will show that the combined parameter clique-core gap plus "respective relaxation parameter" $g + s$ cannot be used to develop FPT-algorithms for the Clique relaxations s-Defective Clique, s-Plex, and s-Club under standard parameterized complexity assumptions. Recall that a given instance (G, k, s) with G being a graph and $k, s \in \mathbb{N}$ is a YES-instance of s-Defective Clique, s-Plex, or s-Club when there exists a set $S \subseteq V(G)$ of order at least k such that: $G[S]$ contains at most s non-edges; $G[S]$ has minimum degree at least $|S| - s$; or $G[S]$ has diameter at most s, respectively. Furthermore, recall that an instance (G, k, ℓ) is a YES-instance of Dense Subgraph when there exists a set $S \subseteq V(G)$ of order at *most* k such that $G[S]$ contains at least $\ell \in \mathbb{N}$ edges.

All of the hardness results will be given with respect to the degeneracy d of a graph, rather than the clique-core gap. This is no problem for our work, as we observe that the clique-core gap of a (non-empty) graph is at most its degeneracy.

Observation 3.1. *For all graphs G it holds that $g(G) \leq d(G)$.*

Thus, any hardness results of a Clique relaxation with respect to d will hold for the parameter g as well.

First, we will prove as an intermediate step that Dense Subgraph is $W[1]$-hard with respect to the combined parameter solution size plus number of contained edges plus degeneracy $k + \ell + d$ by applying techniques from the literature. Next, we will reformulate these results to the strongly related s-Defective Clique problem to show that s-Defective Clique is $W[1]$-complete with respect to $k+s+d$, as well as para-NP-hard with respect to both the individual parameters d and g, where s is the maximum number of missing edges. Finally, we will observe for s-Plex and s-Club from hardness results from the literature that s-Plex is $W[1]$-hard and s-Club is even para-NP-hard with respect to the respective combined parameter $g + s$.

Note that although we study clique relaxations, the parameter $g := d+1-\omega$ depends on the maximum-order ω of a *clique*. As a clique in a graph is also an s-club, s-plex, and s-defective clique for all $s \in \mathbb{N}$, replacing ω with the corresponding maximum-order of a relaxed clique would make the new "relaxed" gap parameters even smaller than the original clique-core gap, hence transferring the hardness results as well.

3.1 Hardness Results for s-Defective Clique

We will show that the Clique relaxation s-Defective Clique is $W[1]$-complete with respect to the combined parameter $k + s + d$. We also show that s-Defective Clique is para-NP-hard with respect to the degeneracy alone. We do this by using the strong relation between ℓ-dense subgraphs and s-defective cliques.

Observation 3.2. *Let G be a graph, $S \subseteq V(G)$ and $\ell \in \mathbb{N}$ be the number of edges in $G[S]$. Then, $S \subseteq V(G)$ is an ℓ-dense subgraph if and only if S is an $(\binom{|S|}{2} - \ell)$-defective clique.*

Proof. Let S be an ℓ-dense subgraph. Then, $G[S]$ contains by definition ℓ edges and misses exactly $\binom{|S|}{2} - \ell$ edges with respect to the complete graph $K_{|S|}$. If S is an $(\binom{|S|}{2} - \ell)$-defective clique, then the other direction holds as well. □

Next, we prove that Dense Subgraph is $W[1]$-hard with respect to $k + \ell + d$, even if the degeneracy is at most two. We mention that Komusiewicz and Sorge [KS15] observed that another reduction of Feige and Seltser [FS97, Theorem 3.1] showed that Dense Subgraph is NP-hard on graphs with degeneracy at most two. However, in their reduction, both k and ℓ depend on n. Next, we show that Dense Subgraph is $W[1]$-hard with respect to the combined parameter $k + \ell + d$, even if d is at most two, by building upon a second reduction from Clique to Dense Subgraph from the literature.

Theorem 3.3 ([RS08, Theorem 20]). Dense Subgraph *is $W[1]$-hard with respect to the solution size k.*

Note that Koana, Komusiewicz, and Sommer [KKS20, Page 8] already mentioned that s-Defective Clique is $W[1]$-hard with respect to k by using Theorem 3.3. However, if one studies the reduction of Raman and Saurabh [RS08, Theorem 20] in more detail, then one can obtain hardness results for the parameters ℓ and d as well.

For Dense Subgraph, a graph on k vertices contains at most $\binom{k}{2}$ edges. Hence, for any Dense Subgraph instance (G, k, ℓ) it holds that $\ell \leq \binom{k}{2}$, or otherwise we have a trivial `NO`-instance. Because ℓ is consequently upper-bounded in $\binom{k}{2}$, it follows from Theorem 3.3 that Dense Subgraph is $W[1]$-hard with respect to the combined parameter $k + \ell$.

Observation 3.4. Dense Subgraph *is $W[1]$-hard with respect to the combined parameter solution size plus number of edges, $k + \ell$.*

Next, we show that Dense Subgraph is $W[1]$-hard with respect to the combined parameter $k+\ell+d$. The observation that leads to this result has already been made by Komusiewicz and Sorge [KS15, Theorem 10] for a variant of Dense Subgraph which is based on the density of the solution. For the sake of completeness, we show that their observation also applies to our Dense Subgraph problem, which is based on the absolute number ℓ of edges in the solution. We also show that each ℓ-dense subgraph in the resulting graph is connected and relatively sparse. We mention that we will use the last two properties for a hardness result of s-Defective Clique in Section 6.1.

Theorem 3.5. Dense Subgraph *is $W[1]$-hard with respect to the combined parameter solution size plus number of edges plus degeneracy, $k+\ell+d$, even if the degeneracy of the resulting graph is at most two, every solution in the resulting graph is connected, and $k \in \Omega(\ell)$.*

Proof. We apply the reduction from the $W[1]$-hard parameterized problem Clique with respect to the solution size k to Dense Subgraph with respect to the solution size k' as presented by Raman and Saurabh [RS08, Theorem 20].

For the sake of completeness, we recall the construction of Raman and Saurabh [RS08, Theorem 20]. Let (G, k) be a Clique instance with $k \geq 3$. Next, we will construct a corresponding Dense Subgraph instance (G', k', ℓ).

First, we fix some arbitrary linear ordering $\preceq$ of $V(G)$. This can be done in linear time. To construct the new graph G', the concept of *subdividing* all edges of G as described in the textbook Diestel [Die16, Section 1.7] is used. This means that we introduce one new vertex w_{uv} for each edge $\{u, v\}$ in G. To be precise, we introduce a new set of vertices $W := \{w_{uv} \mid \{u, v\} \in E; u \preceq v\}$. Now, one replaces each edge $\{u, v\}$ in the original graph G with the corresponding vertex w_{uv} of W and connects the corresponding vertices u and v from $V(G)$ not to each other, but to the vertex w_{uv}, thus "dividing" each edge in G.

All in all, we define the new graph G' with $V(G') := V(G) \cup W$ and $E(G') := \{\{u, w_{u,v}\}, \{v, w_{u,v}\} \mid \{u, v\} \in E(G)\}$. To finish the reduction, let $k' := k + \binom{k}{2}$ and $\ell := 2\binom{k}{2}$. Raman and Saurabh [RS08, Theorem 20] have shown that this reduction is correct and runs in polynomial time.

Due to an observation of Komusiewicz and Sorge [KS15, Theorem 10], each induced subgraph of G' of order exactly $k + \binom{k}{2}$ contains at most $2\binom{k}{2}$ edges. Additionally, they showed that G contains a clique of order exactly k if and only if G' contains an ℓ-dense subgraph of order *exactly* $k' := k + \binom{k}{2}$ with exactly $\ell := 2\binom{k}{2}$ present edges. Hence, we conclude that $k' \in \Omega(\ell)$. Furthermore, we conclude from Raman and Saurabh [RS08, Theorem 20] and Komusiewicz and Sorge [KS15, Theorem 10] that such an ℓ-dense subgraph in G' is always connected.

Komusiewicz and Sorge [KS15, Theorem 10] briefly mentioned that G' has degeneracy at most two. For the sake of completeness, we argue why this is the case. As the vertices in W correspond to the edges in G, we observe that each edge inside G' is incident to some vertex in W. Hence, each vertex in W has degree exactly two, as already mentioned by Raman and Saurabh [RS08, Theorem 20].

All in all, we can construct a degeneracy ordering $\preceq_d$ of $V(G')$ in such a way that each vertex has at most two of its neighbors to the right. We add the vertices from W

at the beginning in an arbitrary ordering to $\preceq_d$. Afterwards, we add the remaining vertices $V(G') \setminus W = V(G)$ from $V(G')$ in an arbitrary ordering to the right of $\preceq_d$.

Now we show that each vertex has at most two neighbors to its right with respect to $\preceq_d$. Because each vertex of W has degree at most two, it can have at most two neighbors to its right. And because all edges in G' are incident to some vertex in W and the remaining vertices $V(G)$ are all to the right of the vertices of W with respect to $\preceq_d$, no vertex of $V(G)$ has a neighbor to its right. Thus, G' has degeneracy at most two.

By applying Observation 3.4 as well, we have shown that Dense Subgraph is $W[1]$-hard with respect to the combined parameter $k+\ell$, even if the degeneracy of the resulting graph is at most two, every solution in G' is connected, and $k' \in \Omega(\ell)$. □

Finally, we can reduce from Dense Subgraph with respect to the combined parameter $k+\ell+d$ to s-Defective Clique with respect to the combined parameter $k+s+d$ by applying Observation 3.2. Hence, an analogous hardness result to Theorem 3.5 follows for s-Defective Clique.

Corollary 3.6. *s-Defective Clique is $W[1]$-hard with respect to the combined parameter solution size plus number of missing edge plus the degeneracy, $k+s+d$, even if the degeneracy of the resulting graph is at most two, every solution in the resulting graph is connected, and $k \in \mathcal{O}(\sqrt{s})$.*

Proof. We reduce a given Dense Subgraph-instance (G, k, ℓ) to the s-Defective Clique-instance $(G' := G, k' := k, s := \binom{k}{2} - \ell)$.

For the sake of completeness, we discuss the potential pitfalls of this reduction.

As mentioned in the proof of Observation 3.4, we can assume that $\ell \le \binom{k}{2}$ and thus s is a non-negative number upper-bounded in some function only depending on $k+\ell$. As we do not change k or the degeneracy of G in any way, it follows from Theorem 3.5 that s-Defective Clique is $W[1]$-hard with respect to the combined parameter $k+s+d$. Furthermore, this reduction is computable in linear time.

Recall that when we directly reduce from the resulting Dense Subgraph-instance of Theorem 3.5, then we can assume that all ℓ-dense subgraphs are connected and of order exactly k'. Due to Observation 3.2 and as we do not change the graph in this reduction in any way, we can assume that all s-defective cliques in G' are connected and of order exactly k'. As we can additionally assume that $k \in \Omega(\ell)$ due to Theorem 3.5, it holds that $s \in \Omega(k^2)$. Finally, as $k' := k$, we conclude that $k' \in \mathcal{O}(\sqrt{s})$. □

Next, this implies that s-Defective Clique is para-NP-hard with respect to d.

Corollary 3.7. *s-Defective Clique is NP-hard on graphs with degeneracy two.*

Note that this result also follows directly from Feige and Seltser [FS97, Theorem 3.1] when applying Observation 3.2 afterwards.

For the sake of completeness, we describe an algorithm that decides s-Defective Clique in linear time for graphs with degeneracy at most one. We observe that graphs with degeneracy at most one are forests. Hence, each connected component with ℓ vertices has $\ell - 1$ edges.

The following greedy algorithm decides an s-Defective Clique instance (G, k, s) when G is a forest. We assume that $k \le n$. First, we sort all connected components inside the forest by their order. Furthermore, let T be the maximum-order tree in this forest and let $S := \emptyset$ be the solution set. While $|S| + |V(T)| < k$, we update S to be $S \cup V(T)$, and T to be the next tree in the sorting. After this loop, we add $k - |S|$ many vertices which are connected in T to S. Finally, we return `YES` if and only if $G[S]$ contains at most s non-edges.

This algorithm is correct, as traversing through the trees in this ordering leads to a graph with the minimum number of missing edges possible. Furthermore, it is easy to see that the algorithm runs in linear time.

Next, we show that s-Defective Clique is contained in $W[1]$ with respect to the combined parameter $k + s + d$. To be precise, we even show the containment for the smaller parameter k. We prove this by using the machine model that characterizes the parameterized complexity class $W[1]$. This characterization was first proven by Chen, Flum, and Grohe [CFG05, Defintion 4, Theorem 16], but we present it as stated by Guo, Niedermeier, and Wernicke [GNW07, Theorem 12]. We refer to Chen, Flum, and Grohe [CFG05, Chapter 3] for the definition of a *nondeterministic RAM program.*

Lemma 3.8. *For a parameterized problem Π, we have $\Pi \in W[1]$ if and only if there exist a computable function f, a polynomial $p(n)$, and a nondeterministic RAM program deciding Π such that for every run of the program on an instance (I, k) (where $|I| = n$),*

1. *it performs at most $f(k) \cdot p(n)$ steps;*
2. *at most the first $f(k) \cdot p(n)$ registers are used;*
3. *at every point of the computation, no register contains numbers strictly greater than $f(k) \cdot p(n)$;*
4. *all nondeterministic steps are among the last $f(k)$ steps.*

We will use Lemma 3.8 to prove that s-Defective Clique is contained in $W[1]$ with respect to the parameter k. Note that our RAM program will be very similar to the RAM program described for Clique with respect to k as sketched by Chen, Flum, and Grohe [CFG05, Section 3]. However, for the sake of completeness, we still present our result as well.

Lemma 3.9. *s-Defective Clique is contained in $W[1]$ with respect to the parameter k, where k is the solution size and s is the maximum number of missing edges.*

Proof. We show that there exists a nondeterministic RAM program as described in Lemma 3.8 deciding s-Defective Clique with respect to the parameter k, which implies Lemma 3.9.

Let (G, k, s) be an s-Defective Clique instance. We assume that $k \leq n$. Furthermore, we assume that the graph G is given in a representation where it takes constant time to check whether any two vertices are adjacent. If this is not the case, then the first step of our algorithm is to construct an adjacency matrix of the graph, which is a graph representation that fulfills this property.

Next, we check whether $s \geq \binom{k}{2}$. If this is the case, then we return `YES`. Afterwards, we compute a potential solution set S by guessing k vertices of G. To verify that $G[S]$ is an s-defective clique of order k, we first check that all vertices in S are pairwise different. This is possible by iterating k times over S and comparing in the i-th iteration the i-th vertex v of S to all vertices u which are right of v in S for all $i \in [k]$, where the ordering of S is given by the ordering of the k registers. Second, we check that there are not too many missing edges in $G[S]$ by counting them. Let ℓ be a counter for the missing edges. We count them by iterating k times over S and comparing in the i-th iteration the i-th vertex v of S to all vertices u which are right of v in S for all $i \in [k]$, where the ordering of S is given by the ordering of the k registers. If v and u are not adjacent in G, then we increase ℓ by one. Furthermore, if at any point $\ell \geq s$ holds, then we abort the RAM program and return `NO`. If we never returned `NO` at any point of the iteration, then we return `YES` in the end.

It is easy to see that our RAM program decides s-Defective Clique. Note that we may return `YES` when $s \geq \binom{k}{2}$, as in this case any arbitrary set of k vertices is an s-defective clique.

Now we analyze the running time and space of this RAM program to show that it satisfies all requirements of Lemma 3.8. Note that a vertex of the graph is uniquely identifiable with $\mathcal{O}(\log n)$ bits, thus storing a vertex inside exactly one register does not violate Lemma 3.8. Hence, constructing an adjacency matrix of the graph is possible in deterministic polynomial time and space. Next, guessing k vertices is possible in $\mathcal{O}(k)$ nondeterministic steps, and the vertices can be stored in overall k registers. Recall that because we used nondeterminism, we afterwards are only allowed to do $g(k)$ more steps to fulfill all requirements of Lemma 3.8 for some computable function g. When we check that all vertices in S are pairwise different, we keep track of the iterations by using a pointer to v and another pointer to u. Furthermore, comparing v and u for equality can be done in constant time. Hence, checking that all vertices in S are pairwise disjoint can be done in deterministic $\mathcal{O}(k^2)$ time and constant additional space. When we check that $G[S]$ does not miss too many edge, we keep track of the iterations again by using a pointer to v and another pointer to u. Furthermore, we assumed that checking the adjacency of two vertices is possible in constant time. Additionally, ℓ is never larger than s, so we are allowed to store the variable ℓ inside one register as $\ell \leq s < \binom{k}{2}$. Hence, checking that $G[S]$ does not miss too many edges can be done in deterministic $\mathcal{O}(k^2)$ time and constant additional space.

With all of this information it is easy to check that our RAM program satisfies all requirements of Lemma 3.8. Hence, Lemma 3.9 holds. □

As $k + s + d$ is larger than k, we conclude from Corollary 3.6 and Lemma 3.9 that s-Defective Clique is $W[1]$-complete with respect to the combined parameter $k+s+d$.

Theorem 3.10. *s-Defective Clique is $W[1]$-complete with respect to the combined parameter $k + s + d$, where k is the solution size, s is the maximum number of missing edges, and d is the degeneracy of the graph.*

3.2 Intractability of the Clique-Core Gap for s-Defective Clique, s-Plex, and s-Club

Recall that s-Defective Clique is $W[1]$-hard with respect to the combined parameter degeneracy plus number of missing edges $d+s$ due to Corollary 3.6. Next, we will present similar results for s-Plex and s-Club from the literature.

First, Koana, Komusiewicz, and Sommer [KKS20, Theorem 3.3] implicitly showed that s-Plex is $W[1]$-hard with respect to $d + s$, where s is the maximum number of neighbors each vertex in the solution is allowed to miss. Second, Hartung et al. [Har+15, Corollary 1] showed that s-Club is NP-hard even if $s = 2$ and $d = 6$, where s is the maximum diameter of the graph induced by the solution set. Due to Observation 3.1, this means that for all three of these Clique relaxations, one has to consider larger parameters than the combined parameter clique-core gap plus the "relaxation parameter" s (or the combined parameter degeneracy plus s, respectively) to develop FPT-algorithms.

Corollary 3.11. *The* Clique *relaxation s-*Club *is para-NP-hard with respect to the combined parameter $g + s$, where $g := d + 1 - \omega$ is the clique-core gap and s is the maximum diameter of the graph induced by the solution set. s-*Defective Clique *and s-*Plex *are $W[1]$-hard with respect to the combined parameter $g + s$, where s is the maximum number of non-edges in the solution, or the maximum number of neighbors each vertex in the solution is allowed to miss, respectively.*

Corollary 3.11 motivates the introduction of "larger" gap parameters for developing FPT-algorithms for Clique relaxations. Hence, in Chapter 4, we will study a much larger gap parameter, that is, $n - \omega$.

Chapter 4

Studying a Larger Gap

In Chapter 3, we have shown that we cannot easily re-formulate the KWB-algorithm of Komusiewicz [Kom11, Proposition 5.4] (re-introduced and implemented by Walteros and Buchanan [WB20, Theorem 1]) for Maximum Clique to clique relaxations with respect to the clique-core gap $g := d + 1 - \omega$, where d is the degeneracy of G, and ω is the order of a maximum clique in G. This is because s-Defective Clique, s-Plex, and s-Club are $W[1]$- or para-NP-hard with respect to the clique-core gap.

The goal of this chapter is to enlarge the "gap" as much as possible in the hope of finding FPT-algorithms for these three Clique relaxations with respect to this "larger gap". Recall that a given instance (G, k, s) with G being a graph and $k, s \in \mathbb{N}$ is a YES-instance of s-Club, s-Plex, or s-Defective Clique when there exists a set $S \subseteq V(G)$ of order at least k such that: $G[S]$ has diameter at most s; $G[S]$ has minimum degree at least $|S| - s$; or contains at most $s \in \mathbb{N}_+$ non-edges, respectively.

Next, recall that for Maximum Clique, the clique-core gap of the input graph G can be seen as the gap between the upper bound $d + 1$ on the order of a clique in G and the actual maximum order ω of a clique in G. In this chapter, we will enlarge the clique-core gap by studying "the largest" upper bound n on the order of a clique in G. Hence we will study the parameter $n - k$ for the three decision Clique relaxations in more detail, where k is the solution size. Note that we consider the solution size rather than the maximum-order of a relaxed clique, because we will study the respective decision problems in this chapter.

First, we will present in Section 4.1 several results from the literature for all three of our Clique relaxations, that is we will note that s-Club is fixed-parameter tractable with respect to $n - k$, and both s-Plex and s-Defective Clique are fixed-parameter tractable with respect to the combined parameter $(n - k) + s$, where s is the respective "relaxation parameter". While presenting these results, we will see that in the literature there exist several problem kernels for s-Plex with respect to $(n - k) + s$ which were constructed by studying the dual problem of s-Plex (which is Bounded-Degree Vertex Deletion). As, to the best of our knowledge, there exist no problem kernels for s-Defective Clique with respect to $(n - k) + s$, this will motivate us to find problem kernels for s-Defective Clique with respect to $(n - k) + s$ by studying the dual problem Partial Vertex Cover of s-Defective Clique in more detail in Sections 4.2 and 4.3.

Note that for a given graph G and integers $\kappa, s \in \mathbb{N}$, the instance (G, κ, s) is a `YES`-instance of Partial Vertex Cover if and only if G contains an s-partial vertex cover S of size at most κ. This means that S covers at least $m - s$ edges, or in other words, there are at most s edges which are not covered by S. Hence, G contains an s-defective clique of order $n - k$ if and only if its complement graph $\overline{G}$ contains an s-partial vertex cover of size $k =: \kappa$.

Observation 4.1. *Let $(G, n - k, s)$ be an instance of s-*Defective Clique*. Then, $(G, n - k, s)$ is a `YES`-instance of s-*Defective Clique *if and only if $(\overline{G}, k, s)$ is a `YES`-instance of* Partial Vertex Cover.

Because of Observation 4.1, we say that Partial Vertex Cover with respect to $\kappa + s$ is the *dual problem* of s-Defective Clique with respect to $(n - k) + s$. In Section 4.2, we will provide a new branching algorithm and a "Buss-like" problem kernel for Partial Vertex Cover with respect to $\kappa + s$. This problem kernel contains $\mathcal{O}(\kappa^2)$ vertices for constant s, and we will provide another, more sophisticated problem kernel with $\mathcal{O}(\kappa)$ vertices for constant s in Section 4.3.

We start by giving an overview of results for our three Clique relaxation when parameterized by $n - k$.

4.1 Overview of Known Results

Next, we give an overview of results for our three Clique relaxation when parameterized by the "large gap" $n - k$.

s-Club. Bourjolly, Laporte, and Pesant [BLP02] proposed a search-tree algorithm for finding a maximum-order s-club in a graph G for all $s \in \mathbb{N}_+$ by branching over two vertices which cannot be part of the same s-club, that is, they have distance at least $s+1$, until the resulting instance is an s-club. Schäfer et al. [Sch+12, Theorem 5] proved that this branching algorithm can be used to decide whether (G, k, s) is a `YES`-instance of s-Club in $\mathcal{O}^*(2^{n-k})$ time by branching at most $n - k$ times.

Hence, s-club is fixed-parameter tractable with respect to $n - k$.

s-Plex and Bounded-Degree Vertex Deletion. Many results for s-Plex with respect to $n - k$ have been found by studying the dual problem Bounded-Degree Vertex Deletion of s-Plex, hence we quickly introduce Bounded-Degree Vertex Deletion [NRT05].

For a given graph G and integers $\kappa, d \in \mathbb{N}$, the instance (G, κ, d) is a `YES`-instance of Bounded-Degree Vertex Deletion if and only if G contains a bdd-d-set S of size at most κ. This means that $G - S$ has maximum degree at most d. Hence, G contains an s-plex of order $n - k$ if and only if its complement graph $\overline{G}$ contains a bdd-$(s - 1)$-set of size $k =: \kappa$. Thus, we say that Bounded-Degree Vertex Deletion with respect to $\kappa + d$ is the *dual problem* of s-Plex with respect to $(n - k) + s$. The following results were all originally proven for Bounded-Degree Vertex Deletion.

Fellows et al. [Fel+11, Theorem 2] showed that s-Plex is $W[2]$-complete with respect to $n - k$.

On the positive side, Hartung et al. [Har+15, Theorem 6] showed indirectly that for all $s \in \mathbb{N}_+$ one can find a maximum-order s-plex of a graph G by branching over $s+1$ vertices which cannot be part of the same s-plex, that is, one of the vertices is non-adjacent to all the other s other vertices, until the resulting instance is an s-plex. They also proved that this branching algorithm can be used to decide whether (G, k, s) is a `YES`-instance of s-Plex in $\mathcal{O}^*((s+1)^{n-k})$ time by branching at most $n-k$ times. Furthermore, for constant s, Fellows et al. [Fel+11, Theorem 1] provided a quasi-linear–vertex problem kernel for s-Plex with respect to $n-k$ which is computable in $\mathcal{O}(n^4 \cdot m)$ time, and Moser, Niedermeier, and Sorge [MNS12, Theorem 1] provided a "Buss-like" quadratic-vertex problem kernel with respect to $n-k$ which is computable in quadratic time.

s-Defective Clique and Partial Vertex Cover. Some results for s-Defective Clique can be transferred from the its dual problem Partial Vertex Cover.

Guo, Niedermeier, and Wernicke [GNW07, Theorem 14] and Cai [Cai08, Theorem 2.5] showed that s-Defective Clique is $W[1]$-complete with respect to $n-k$.

On the positive side, Raman and Saurabh [RS08, Theorem 10] showed that for all $s \in \mathbb{N}_+$ one can find a maximum-order s-defective clique of a graph G by branching over a missing *edge* into three different cases, until the resulting instance is an s-defective clique. They also proved that this branching algorithm can be used to decide whether (G, k, s) is a `YES`-instance of s-Plex in $\mathcal{O}^*(3^{(n-k)+s})$ time by branching at most $(n-k)+s$ times.

To the best of our knowledge, no branching algorithm over a *set of vertices* was considered for s-Defective Clique before, in contrast to s-Club and s-Plex. As our goal in this thesis is to find a *framework* for Clique relaxations, it is of high interest to find such a branching algorithm for s-Defective Clique too. Furthermore, it is of interest to provide two problem kernels for s-Defective Clique with respect to $(n-k)+s$ of "similar sizes" as the ones for s-Plex, as each s-defective clique is an $s+1$-plex as observed by Trukhanov et al. [Tru+13, Section 3.1] and Shirokikh [Shi13, Proposition 2.7]. As all three of these results for s-Plex were found by studying the dual problem Bounded-Degree Vertex Deletion, we next will concentrate on the dual problem Partial Vertex Cover of s-Defective Clique, and consider the parameterization $\kappa + s$. For the sake of completeness, we mention that under standard complexity assumptions it is unlikely that Partial Vertex Cover is fixed-parameter tractable with respect to κ or s alone. This is because Partial Vertex Cover is $W[1]$-complete with respect to κ [Cai08; GNW07], and para-NP-hard with respect to $s = 0$, as this special case coincides with the NP-hard Vertex Cover problem [Kar72].

4.2 Basic FPT-Results for Partial Vertex Cover

Recall that Raman and Saurabh [RS08, Theorem 10] provided an $\mathcal{O}^*(3^{\kappa+s})$-time algorithm for Partial Vertex Cover by branching over a missing edge. In the following, we will provide an $\mathcal{O}^*((2 \cdot (s+1))^{\kappa})$-time algorithm for Partial Vertex Cover by branching over a set of vertices. Afterwards, we will construct a "Buss-like" problem kernel as well.

Branch over set of vertices. For the standard Vertex Cover problem, it is known that for each edge $\{u, v\}$ of a graph G, every vertex cover of G has to contain u or v, as otherwise the edge $\{u, v\}$ is not covered. One may use this observation to construct an algorithm for Vertex Cover running in $\mathcal{O}(2^\kappa \cdot (n + m))$ time as explained in the textbook of Mehlhorn [Meh84, Chapter 6]. Next, we lift this idea to Partial Vertex Cover by observing that if G contains strictly more than s edges, then at least one of the endpoints of these edges has to be part of any s-partial vertex cover S of G, as otherwise there are strictly more than s edges in G not covered by S, contradicting the definition of an s-partial vertex cover. This observation was already made by Guo et al. [Guo+11, Theorem 2] in the context of a Cluster Editing variant in a slightly different form.

Observation 4.2. *Let G be a graph and $s \in \mathbb{N}$. If G contains at least $s + 1$ edges, then every s-partial vertex cover of G has to contain at least one endpoint of these edges.*

Observation 4.2 leads to a simple depth-bounded search tree algorithm computing an s-partial vertex cover of order at most $\kappa \in \mathbb{N}$ when there exists such a set inside a graph G, or `NONE` otherwise. If κ is non-positive, then we return `NONE`. If G contains at most s edges, then we return the empty set. Otherwise, we branch over all endpoints of $s+1$ edges inside G, and assume that the current endpoint is part of an s-partial vertex cover of order at most κ, thus removing the vertex from the graph and decreasing κ by one. If one of the branches returns an s-partial vertex cover, then we return this s-partial vertex cover together with the selected endpoint of this branch, otherwise we return `NONE`.

This algorithm is correct due to Observation 4.2. Note that $s + 1$ edges have at most $2 \cdot (s+1)$ unique endpoints, and we branch at most κ times. Furthermore, counting the number of edges in G, constructing a set of $s + 1$ edges of G, and constructing the subgraphs is possible in linear time. Hence, we arrive at the following result.

Proposition 4.3. *Let G be a graph. An s-partial vertex cover of order at most $\kappa \in \mathbb{N}$ is computable in $\mathcal{O}((2 \cdot (s + 1))^\kappa \cdot (n + m))$ time, where $s \in \mathbb{N}$ is the maximum number of edges not covered.*

Note that Proposition 4.3 is strongly related to the algorithm of Guo et al. [Guo+11, Theorem 3] for deciding a Cluster Editing variant.

Next, we show that in different circumstances, both the $\mathcal{O}^*(3^{\kappa+s})$-time algorithm of Raman and Saurabh [RS08, Theorem 10] (denoted with ***`Edge-Alg`***) and the algorithm of Proposition 4.3 (denoted with *`Vertex-Alg`*) could perform better than their respective other algorithm. First, note that for constant κ, `Vertex-Alg` runs in polynomial time, while `Edge-Alg` runs in $\mathcal{O}^*(3^s)$ time. In contrast, as $\mathcal{O}((2 \cdot (s + 1))^\kappa) \subseteq 3^{\mathcal{O}(\log(s) \cdot \kappa)}$, if κ and s are unbounded, then `Edge-Alg` might perform better in practice than `Vertex-Alg`. This is because in the exponent of the running time of `Edge-Alg`, there exists an additive dependency between κ and s, while `Vertex-Alg` has a multiplicative dependency between them.

Next, we present two problem kernels for Partial Vertex Cover with respect to $\kappa + s$.

Buss-like problem kernel. We briefly discuss a problem kernel for Partial Vertex Cover with respect to $\kappa + s$ with $\mathcal{O}(\kappa \cdot (\kappa + s))$ vertices. This is meant as a warm-up for the more involved problem problem kernel in Section 4.3, where we will exploit the idea of removing high-degree vertices even more.

Like the data reduction rules of Buss and Goldsmith [BG93] for Vertex Cover, we present a data reduction rule for isolated vertices as well as a data reduction rule for "high"-degree vertices. Furthermore, we will show that these data reduction rules are applicable in linear time.

Reduction Rule 4.2.1. *Let (G, κ, s) be an instance of* Partial Vertex Cover. *If there is a vertex $v \in V(G)$ with $\deg(v) = 0$, then return $(G - v, \kappa, s)$.*

We observe that removing isolated vertices from a Partial Vertex Cover instance is safe. Furthermore, removing all isolated vertices from G can be done in $\mathcal{O}(n+m)$ time by iterating once over the graph.

Lemma 4.4. *Reduction Rule 4.2.1 is safe and it is exhaustively applicable in linear time.*

Next, we observe that a vertex v of some graph G with at least $\kappa+s+1$ neighbors has to be part of every s-partial vertex cover S of size at most κ for some $\kappa, s \in \mathbb{N}$. If v would not be a part of S, then S would cover at most κ incident edges of v, leaving $s+1$ edges in G not covered.

Observation 4.5. *Let G be a graph and $\kappa, s \in \mathbb{N}$. If there exists a vertex $v \in V(G)$ of degree at least $\kappa + s + 1$, then every s-partial vertex cover of G of size at most κ contains v.*

Next, we formulate a "high-degree" data reduction rule based on Observation 4.5.

Reduction Rule 4.2.2. *Let (G, κ, s) be an instance of* Partial Vertex Cover. *If there is a vertex $v \in V(G)$ with $\deg(v) \geq \kappa + s + 1$, then return $(G - v, \kappa - 1, s)$.*

Lemma 4.6. *Reduction Rule 4.2.2 is safe and it is exhaustively applicable in linear time.*

Proof. The correctness of Reduction Rule 4.2.2, that is the statement (G, κ, s) is a YES-instance of Partial Vertex Cover if and only if $(G - v, \kappa - 1, s)$ is a YES-instance of Partial Vertex Cover, holds trivially due to Observation 4.5.

Next, we show how to exhaustively apply Reduction Rule 4.2.2 in $\mathcal{O}(n + m)$ time. We assume that G is given as an adjacency list. Next, we sort the vertices of G by their degree in an ascending ordering, which can be done with a bucket-sort approach in linear time. We say that v_i is the vertex which is represented by the i-th entry in the adjacency list with $i \in [n]$.

First, we construct an array of length n where the i-th entry contains the degree of v_i for $i \in [n]$. This can be done by traversing the adjacency list once in $\mathcal{O}(n + \sum_{i\in[n]} \deg(v_i)) = \mathcal{O}(n+m)$ time. This array is the *label array*. Furthermore, we initialize the variable κ' with κ.

Next, we iterate over the label array with $i \in (n, n-1, \ldots, 1)$, starting at the label of v_n. If the current label of the vertex v_i is at least $\kappa' + s + 1$, then v_i has to be part of every s-partial vertex cover of size κ due to Observation 4.5. Thus, we decrease κ' by one, and decrease every label of each neighbor of v_i by one, as we will remove v_i later on. Otherwise, the current label of the vertex v_i is less than $\kappa' + s + 1$, and we stop the whole iteration and define the variable p to be $i + 1$. As we remove every vertex at most once, the whole iteration takes overall $\mathcal{O}(n + \sum_{i \in [n]} \deg(v_i)) = \mathcal{O}(n + m)$ time.

As the vertices $v_p, v_{p+1}, \ldots, v_n$ all have to be part of every s-partial vertex cover of size at most κ, we will construct the induced subgraph G^* of G on the vertex set $\{v_1, v_2, \ldots, v_{p-1}\}$. First, we remove for all vertices v_i all neighbors v_j with $j \geq p$ in overall $\mathcal{O}(n + \sum_{i \in [n]} \deg(v_i)) = \mathcal{O}(n + m)$ time, where $i \in [n], v_j \in N(v_i)$. Afterwards, we say that the array of the adjacency list of G^* ends at position $p - 1$, which can be done in constant time. Afterwards, we return (G^*, κ', s). All in all, exhaustively applying Reduction Rule 4.2.2 takes linear time. □

Next, we show that these two data reduction rules are enough to find an upper bound on the number of vertices in G with respect to $\kappa + s$. Note that we assume κ to be non-negative, as there exists no s-partial vertex cover of negative size.

Proposition 4.7. *Let (G, κ, s) be an instance of* Partial Vertex Cover *such that Reduction Rules 4.2.1 and 4.2.2 are not applicable anymore. If G contains strictly more than $\kappa \cdot (\kappa + s) + 2s$ vertices, then (G, κ, s) is a trivial* `NO`*-instance of* Partial Vertex Cover.

Proof. We observe that G does not contain any isolated vertices due to Reduction Rule 4.2.1, and each vertex in G is incident to at most $\kappa + s$ edges due to Reduction Rule 4.2.2, and there are at most s edges with at most $2s$ unique endpoints which are not covered by any s-partial vertex cover of G. Thus, we conclude that if G contains strictly more than $\kappa \cdot (\kappa + s) + 2s$ vertices, then (G, κ, s) is a `NO`-instance of Partial Vertex Cover. □

All in all, we compute a problem kernel for Partial Vertex Cover with at most $\kappa \cdot (\kappa + s) + 2s$ vertices by exhaustively applying Reduction Rules 4.2.1 and 4.2.2 in linear time, as well as counting the vertices in the resulting graph to apply Proposition 4.7 in linear time.

Theorem 4.8. Partial Vertex Cover *admits a problem kernel with at most $\kappa \cdot (\kappa + s) + 2s$ vertices, where κ is the solution size and s is the maximum number of edges not covered. This problem kernel is computable in linear time.*

Note that a similar approach was used by Moser, Niedermeier, and Sorge [MNS12, Theorem 1] to provide a similar problem kernel for Bounded-Degree Vertex Deletion with respect to the combined parameter solution size plus "relaxation parameter".

We observe that Partial Vertex Cover admits a problem kernel with $\mathcal{O}(\kappa^2)$ vertices for constant s due to Theorem 4.8. In the next section, we will present a problem kernel for Partial Vertex Cover that for constant s admits a problem kernel with $\mathcal{O}(\kappa)$ vertices.

4.3 Linear Vertex Kernel for Partial Vertex Cover

In this section, we will show the following problem kernel.

Theorem 4.9. PARTIAL VERTEX COVER *admits a problem kernel with strictly less than $(s+2)\cdot(\kappa+s)$ vertices, where κ is the solution size and s is the maximum number of edges not covered. This problem kernel is computable in $\mathcal{O}(\kappa\cdot(n+m)^2)$ time.*

Note that this means that for constant s, PARTIAL VERTEX COVER admits a problem kernel with $\mathcal{O}(\kappa)$ vertices.

To prove Theorem 4.9, we use techniques known in the context of linear programming as well as the *expansion lemma* from [PR05, Corollary 8.1], which we will recall in Section 4.3.1. Afterwards, we will prove Theorem 4.9 by partitioning the vertices of a graph in three sets and find an upper bound on the number of vertices inside each set in Sections 4.3.2 and 4.3.3. In Section 4.3.4, we will put everything together to prove Theorem 4.9 and show that our analysis is tight, that is we need new data reduction rules to find a problem kernel with less vertices for PARTIAL VERTEX COVER with respect to the combined parameter $\kappa+s$.

4.3.1 Fundament

In the following, we will recall techniques known in the context of linear programming as well as the *expansion lemma*, which we will apply to find an upper bound on the number of vertices of the resulting graph in Sections 4.3.2 and 4.3.3.

Linear Programming

To compute our problem kernel as described in Theorem 4.9, we present the standard VERTEX COVER problem as both an INTEGER LINEAR PROGRAMMING (ILP) and a LINEAR PROGRAMMING (LP) formulation as stated in the textbook Cygan et al. [Cyg+15, Section 2.5]. Both formulations are folklore.

Given a graph G, we say that the following ILP formulation is the ILPVC*(G) of G*:

$$\begin{aligned} &\min \sum_{v\in V(G)} x_v && \\ &\text{subject to } x_u + x_v \geq 1 && \forall \{u,v\}\in E(G) \\ &0\leq x_v\leq 1 && \forall v\in V(G) \\ &x_v\in\{0,1\} && \forall v\in V(G) \end{aligned}$$

Let G be graph. If the variables $(x_v)_{v\in V(G)}$ satisfy all constraints of ILPVC(G), then $(x_v)_{v\in V(G)}$ is a *feasible solution* of ILPVC(G). If the value $\sum_{v\in V(G)} x_v$ of a feasible solution $(x_v)_{v\in V(G)}$ of ILPVC(G) is the minimum of all values of the feasible solutions of ILPVC(G), then $(x_v)_{v\in V(G)}$ is an *optimal solution* of ILPVC(G). The value of any optimal solution for ILPVC(G) is the *optimal value* of ILPVC(G).

Since it is NP-hard to compute the optimal value of an ILPVC(G) instance, we relax the ILP formulation of VERTEX COVER to an LP formulation, which is solvable in polynomial time [Kha80].

Given a graph G, we say that the following LP formulation is the LPVC*(G) of G*:

$$\min \sum_{v\in V(G)} x_v$$

$$\text{subject to } x_u + x_v \geq 1 \qquad \forall \{u,v\} \in E(G) \tag{4.1}$$

$$0 \leq x_v \leq 1 \qquad \forall v \in V(G) \tag{4.2}$$

Let G be a graph. We define the notation of ILPVC(G) analogously for LPVC(G). Additionally, if all values x_v of a feasible solution $(x_v)_{v\in V(G)}$ of LPVC(G) are 1, $1/2$, or 0, then $(x_v)_{v\in V(G)}$ is called *half-integral*. For a half-integral feasible solution, we define the partitioning $V(G) =: V_1(G,(x_v)_{v\in V(G)}) \uplus V_{1/2}(G,(x_v)_{v\in V(G)}) \uplus V_0(G,(x_v)_{v\in V(G)})$, where

- $V_1(G,(x_v)_{v\in V(G)}) := \{v \in V(G) \mid x_v = 1\}$,
- $V_{1/2}(G,(x_v)_{v\in V(G)}) := \{v \in V(G) \mid x_v = 1/2\}$, and
- $V_0(G,(x_v)_{v\in V(G)}) := \{v \in V(G) \mid x_v = 0\}$.

If the half-integral solution $(x_v)_{v\in V(G)}$ is clear from the context, then we will omit the argument $(x_v)_{v\in V(G)}$ from these functions.

As an optimal solution for ILPVC(G) is also a feasible solution for LPVC(G), the optimal value of LPVC(G) is a lower bound on the minimum size of a vertex cover with respect to G.

Observation 4.10 ([NT75, Section 3])**.** *Let G be a graph. Then, the optimal value of* LPVC*(G) is a lower bound on the minimum size of a vertex cover of G.*

Next, it is easy to see that $N(V_0(G)) \subseteq V_1(G)$, because if an edge of G is incident to a vertex of $V_0(G)$, then it has to be incident to a vertex of $V_1(G)$ as well. Otherwise, Inequality 4.1 would not hold for this edge.

Observation 4.11 ([NT75])**.** *Let G be a graph and $(x_v)_{v\in V(G)}$ a half-integral optimal solution of* LPVC*(G). It holds that $N(V_0(G)) \subseteq V_1(G)$.*

Nemhauser and Trotter [NT75] showed that for any graph G, there exists a half-integral optimal solution $(x_v)_{v\in V(G)}$ of LPVC(G). Furthermore, they showed that there also exists a corresponding minimum-size vertex cover S of G for $(x_v)_{v\in V(G)}$ such that $V_1(G) \subseteq S$. Hence, $S \cap V_0(G) = \emptyset$ holds due to Observation 4.11. Thus, this half-integral optimal solution gives us a 2κ-vertex problem kernel for the Vertex Cover problem as observed by Chen, Kanj, and Jia [CKJ01], because we only have to consider the subgraph of G induced by the vertex set $V_{1/2}(G)$ any longer and reduce the solution size κ by $|V_1(G)|$. One observes that 2κ is an upper bound on $\left|V_{1/2}(G)\right|$ by applying Observation 4.10, although Nemhauser and Trotter [NT75] proved that this upper-bound exists in a slightly different form.

Note that Cygan et al. showed that a half-integral optimal solution for LPVC(G) is computable with the approach of Nemhauser and Trotter [NT75, Theorem 3], using an adapted version of the algorithm of Hopcroft and Karp [HK73], in $\mathcal{O}(m\sqrt{n})$ time.

Theorem 4.12 ([Cyg+15, Proposition 2.23])**.** Vertex Cover *admits a problem kernel with at most 2κ vertices. This problem kernel is computable in $\mathcal{O}(m\sqrt{n})$ time.*

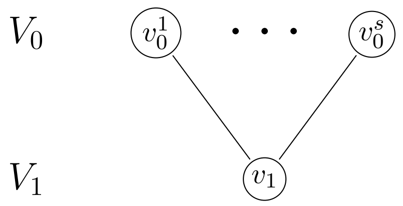


Figure 4.1: An illustration of the graph $K_{1,s}$ with $s \geq 1$ and the vertex partitions given by a half-integral optimal solution $(x_v)_{v \in V(K_{1,s})}$ for LPVC($K_{1,s}$). As $K_{1,s}$ is a star with v_1 as its center and at least one leaf, there always exists an optimal solution for LPVC($K_{1,s}$) with assignments $x_{v_1} := 1$ and $x_{v_0^i} := 0$ for all $s \in \mathbb{N}_+, i \in [s]$. Hence, for each s, there exists a minimum-size vertex cover S of $K_{1,s}$ that only contains v_1. However, for each $s \in \mathbb{N}_+$ every minimum-size s-partial vertex cover of $K_{1,s}$ is empty, because $K_{1,s}$ only contains s edges. Thus, an s-partial vertex cover does not have to cover any edge of $K_{1,s}$. Hence, it is possible that no minimum-size s-partial vertex cover of a graph G contains all vertices of $V_1(G)$.

In contrast to Vertex Cover, we observe that we cannot add the set $V_1(G)$ completely to the solution set of Partial Vertex Cover. The problem is that a minimum s-partial vertex cover for a graph G does not necessarily cover all edges inside the graph. An example for such a situation is given in Figure 4.1. Hence, we have to find an upper bound on the size of $|V_1(G)|$ for Partial Vertex Cover as well. In Section 4.3.2, we will find an upper bound on the combined size $|V_1(G)| + \left|V_{1/2}(G)\right|$ by applying Observation 4.10 to Partial Vertex Cover. However, to find an upper bound on the size of $V_0(G)$ in Section 4.3.3, we also use the *expansion lemma*, which we will discuss next.

Expansion Lemma

To compute an upper bound on the size of $V_0(G)$, we use the *expansion lemma* of [PR05, Corollary 8.1], which builds upon the concept of a *q-expansion* as described in the textbook of Cygan et al. [Cyg+15, Section 2.3, Section 2.4, Lemma 2.18]. We briefly mention that a q-expansion induces a more restricted *crown*.

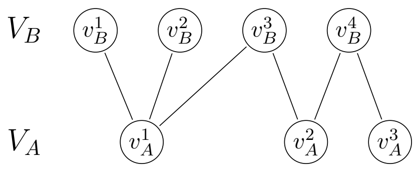
Next, we define the necessary notation to understand the expansion lemma. Let G be a graph, $U, W \subseteq V(G)$ be two disjoint vertex sets, and $p \in \mathbb{N}$. Furthermore, let $M \subseteq E(G)$ be an edge set such that each edge has one endpoint in U and one in W. If exactly p vertices of W are an endpoint of some edge in M, then we say that M *saturates exactly p vertices in W*. Now, we define the concept of a q-expansion.

Definition 4.13 ([PR05, Definition 8.1])**.** Let H be a bipartite graph with vertex bipartition $(V_A(H), V_B(H))$, $M \subseteq E(H)$, and $q \in \mathbb{N}_+$. If

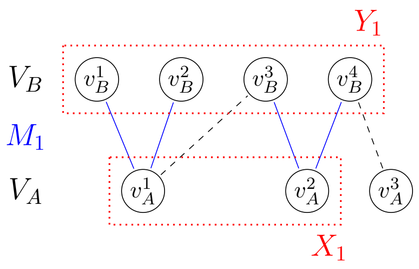
- every vertex of $V_A(H)$ is incident to exactly q edges of M, and
- M saturates exactly $q \cdot |V_A(H)|$ vertices in $V_B(H)$,

then we say that M is a *q-expansion of $V_A(H)$ into $V_B(H)$*.

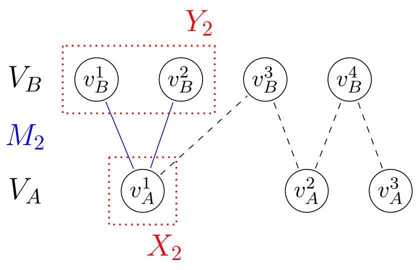
See Figures 4.2b and 4.2c for two examples of such q-expansions M_1 and M_2.



(a) A bipartite graph H



(b) A 2-expansion M_1 of X_1 into Y_1. The sets X_1 and Y_1 do not fulfill the requirements of Lemma 4.14.



(c) A 2-expansion M_2 of X_2 into Y_2. The sets X_2 and Y_2 fulfill the requirements of Lemma 4.14.

Figure 4.2: Figures 4.2a to 4.2c all show the same bipartite graph H. The (blue) edge set M_1 is a 2-expansion of X_1 into Y_1. Note that the edge $\{v_A^1, v_B^3\}$ cannot extend the 2-expansion M_1, as v_A^1 has to be incident to *exactly* two edges in M_1. Furthermore, the (blue) edge set M_2 is a 2-expansion of X_2 into Y_2 as well.

Now, we have presented everything to understand the expansion lemma. It states that we can find a q-expansion in a "large" bipartite graph in quadratic time.

Lemma 4.14 (Expansion Lemma [PR05, Corollary 8.1]). *Let H be a bipartite graph and $q \in \mathbb{N}_+$. If*

- $|V_B(H)| \geq q \cdot |V_A(H)|$, *and*
- *there are no isolated vertices in $V_B(H)$,*

then there exist non-empty vertex sets $X \subseteq V_A(H)$ and $Y \subseteq V_B(H)$ such that

- *there is a q-expansion of X into Y, and*
- *no vertex in Y has a neighbor outside X, that is, $N(Y) \subseteq X$.*

Furthermore, appropriate sets X and Y are computable in $\mathcal{O}((n+m)^2)$ time.

We observe that the vertex sets X and Y are more restricted than a q-expansion. For example, we observe that the sets X_2 and Y_2 of Figure 4.2c fulfill the requirements of the sets X and Y as specified in Lemma 4.14 for H and $q := 2$. However, this is not the case for X_1 and Y_1 of Figure 4.2b, as $N(Y_1) \not\subseteq X_1$, although M_1 is a 2-expansion.

We mention that we will use this additional property of X and Y to find an upper bound on the size of $V_0(G)$ in Section 4.3.3.

Note that Thomassé [Tho10, Theorem 2.4] also showed how to compute X and Y in polynomial time in the case of $q = 2$.

Next, we start to compute our problem kernel by finding an upper bound on the sizes of the sets $V_1(G)$ and $V_{1/2}(G)$.

4.3.2 Upper Bound on $|V_1(G)|$ and $\left|V_{1/2}(G)\right|$

We start by relating the minimum size of a vertex cover in a graph G to the minimum size of an s-partial vertex cover in G. This helps us to mirror the approach of Observation 4.10 towards the minimum size of an s-partial vertex cover of G.

Lemma 4.15. *Let G be a graph. For any given $\kappa, s \in \mathbb{N}$, if G contains an s-partial vertex cover of size at most κ, then G contains a vertex cover of size at most $\kappa + s$.*

Proof. Assume towards a contradiction that G contains an s-partial vertex cover S of size at most κ, but G does not contain a vertex cover of size at most $\kappa + s$.

By the definition of an s-partial vertex cover, $G - S$ contains at most s edges. Thus, the s-partial vertex cover S can be extended to a vertex cover S' of G by adding one arbitrary endpoint of each edge to the set S. The set S' therefore has size $|S| + |E(G - S)| \leq \kappa + s$. Thus, S' is a vertex cover of G of size at most $\kappa + s$, which is a contradiction to the assumption. Hence, the original assumption of Lemma 4.15 holds. $\square$

Recall that due to Observation 4.10, the value of an optimal solution for LPVC(G) is a lower bound on the minimum size of a vertex cover of a graph G. Next, we use the contrapositive of Lemma 4.15 to relate the optimal value of LPVC(G) and the minimum size of an s-partial vertex cover of G.

Corollary 4.16. *Let G be a graph and $\kappa, s \in \mathbb{N}$. If the optimal value of* LPVC*(G) exceeds $\kappa + s$, then G does not contain an s-partial vertex cover of size at most κ.*

Next, we will use Corollary 4.16 to construct a corresponding data reduction rule for the Partial Vertex Cover problem.

Reduction Rule 4.3.1. *Let (G, κ, s) be an instance of* Partial Vertex Cover *with κ being the solution size and s the maximum number of edges not covered. Furthermore, let $(x_v)_{v \in V(G)}$ be a half-integral optimal solution of* LPVC*(G). If the inequality $|V_1(G)| + 1/2 \cdot \left|V_{1/2}(G)\right| \leq \kappa + s$ does not hold, then return a trivial* NO*-instance of* Partial Vertex Cover*.*

Lemma 4.17. *Reduction Rule 4.3.1 is safe and applicable in linear time.*

Proof. We prove the correctness of Reduction Rule 4.3.1, that is we show (G, κ, s) is a YES-instance of Partial Vertex Cover if and only if (G', κ', s') is a YES-instance of Partial Vertex Cover. We assume that Reduction Rule 4.3.1 is actually applied, as otherwise the correctness holds trivially.

"$\Rightarrow$": Because (G, κ, s) is a YES-instance of Partial Vertex Cover, the value of $\sum_{v \in V(G)} x_v$ is at most $\kappa + s$. Since $\sum_{v \in V(G)} x_v = |V_1(G)| + 1/2 \cdot |V_{1/2}(G)| \leq \kappa + s$ holds due to Corollary 4.16, we conclude that the inequality of Reduction Rule 4.3.1 is fulfilled and Reduction Rule 4.3.1 returns the original YES-instance of Partial Vertex Cover, thus (G', κ', s) is a YES-instance of Partial Vertex Cover.

"$\Leftarrow$": Because (G', κ', s') is a YES-instance of Partial Vertex Cover, Reduction Rule 4.3.1 has not changed the original (YES-)instance of Partial Vertex Cover. Thus, (G, κ, s) is a YES-instance of Partial Vertex Cover.

For the running time, note that we assume that $(x_v)_{v \in V(G)}$ is part of the input. Thus, counting the number of vertices in $V_1(G)$ and $V_{1/2}(G)$ as well as computing a trivial NO-instance of Partial Vertex Cover can be done in linear time. □

By the inequality inside of Reduction Rule 4.3.1, two upper bounds on the sizes of the sets $V_1(G)$ and $V_{1/2}(G)$ follow.

Lemma 4.18. *Let (G, κ, s) be an instance of* Partial Vertex Cover *and $(x_v)_{v \in V(G)}$ a half-integral optimal solution of* LPVC*(G) such that Reduction Rule 4.3.1 is not applicable anymore. Then, the inequality $|V_1(G)| + 1/2 \cdot |V_{1/2}(G)| \leq \kappa + s$ holds.*

We will study the relation of the sizes of the sets $V_1(G)$ and $V_{1/2}(G)$ in more detail at the end of this section. Next, we relate the sizes of the sets $V_1(G)$ and $V_0(G)$ to find an upper bound on the size of $V_0(G)$.

4.3.3 Upper Bound on $|V_0(G)|$

In the following, we show that if G contains an $(s+1)$-expansion, then one can remove some vertices inside G in quadratic time. Next, we will observe that if $V_0(G)$ is "large enough", then the expansion lemma is applicable to G and thus we will find an $(s+1)$-expansion for G. All in all, we will find an upper bound on the size of $V_0(G)$.

Using the Expansion Lemma

We show that if a graph G contains a bipartite subgraph H which is applicable to the expansion lemma, then there exists a minimum-size s-partial vertex cover of G which contains the vertex set X completely and is disjoint from the vertex set Y, with X and Y as specified in the expansion lemma. Thus, G contains an s-partial vertex cover of size at most κ if and only if $G - (X \cup Y)$ contains an s-partial vertex cover of size at most $\kappa - |X|$ for some $\kappa, s \in \mathbb{N}$. To prove this, we introduce some additional notation.

Let H be a bipartite graph with vertex bipartition $(V_A(H), V_B(H))$, and $q \in \mathbb{N}_+$ such that there exists a q-expansion M of $V_A(H)$ into $V_B(H)$. For convenience, denote $X := V_A(H)$ and $Y := V_B(H)$. For a vertex $x \in X$, we define $N_{M,Y}(x) \subseteq Y$ to be the set of all vertices in Y adjacent to x over edges in M, formally, $N_{M,Y}(x) := \{y \in Y \mid \{x, y\} \in M\}$. We also define $N_{M,Y}[x] := N_{M,Y}(x) \cup \{x\}$. Finally, we say that $M_Y(x) \subseteq M$ is the set of all edges in M incident to x, formally, $M_Y(x) := \{\{x, y\} \in M \mid y \in Y\}$. If the sets X and Y are clear from the context, then we omit their subscripts. For example, for the sets M_1, X_1, and Y_1 of Figure 4.2b, the sets $N_{M_1}(x)$ are $N_{M_1}(v_A^1) := \{v_B^1, v_B^2\}$ as well as $N_{M_1}(v_A^2) := \{v_B^3, v_B^4\}$, and the sets $M_1(x)$ are $M_1(v_A^1) := \{\{v_A^1, v_B^1\}, \{v_A^1, v_B^2\}\}$ as well as $M_1(v_A^2) := \{\{v_A^2, v_B^3\}, \{v_A^2, v_B^4\}\}$.

Note that, due to the definition of a q-expansion, for all $x \in X$ it holds that $|M(x)| = q$. Thus, as all edges in $M(x)$ are only incident to vertices in $N_M[x]$, if $q := s + 1$, then every s-partial vertex cover of G has to contain at least one vertex of $N_M[x]$ due to Observation 4.2.

Proposition 4.19. *Let H be a bipartite graph and $s \in \mathbb{N}$ such that Lemma 4.14 is applicable to H and $q := s + 1$. Next, let $X \subseteq V_A(H), Y \subseteq V_B(H)$ be the vertex sets as specified in Lemma 4.14, and let M be an $(s+1)$-expansion of X into Y. Then, every s-partial vertex cover of H contains at least one vertex from $N_M[x]$ for all $x \in X$.*

Next, we lift Proposition 4.19 to be applicable to a graph G which contains a bipartite subgraph $H \subseteq G$ for which Lemma 4.14 is applicable.

Lemma 4.20. *Let G be a graph, $H \subseteq G$ be a bipartite subgraph of G, and $s \in \mathbb{N}$. Assume that $N_G(V_B(H)) \subseteq V_A(H)$. Furthermore, assume Lemma 4.14 is applicable to H with $q := s + 1$ and let X and Y be the vertex sets as specified in Lemma 4.14. Then, there exists a minimum-size s-partial vertex cover of G which contains X completely and is disjoint from Y.*

Proof. Let S be a minimum-size s-partial vertex cover of G. We claim that $S^* := (S \setminus Y) \cup X$ fulfills all requirements of Lemma 4.20. To show this, we prove the following three claims:

1. $S^* \cap (X \cup Y) = X$,
2. $|S| \geq |S^*|$, and
3. $\{u, v\} \in E(G)$ is covered by $S \implies \{u, v\}$ is covered by S^*.

These three conditions together imply that S^* is a minimum-size s-partial vertex cover of G such that $S^* \cap (X \cup Y) = X$. Note that S^* is an s-partial vertex cover of G because of the third condition. Hence, if we show that S^* satisfies all three conditions, then Lemma 4.20 holds.

The first condition $S^* \cap (X \cup Y) = X$ holds by the definition of S^*.

Next, we show that $|S| \geq |S^*|$. There exists a q-expansion $M \subseteq E(H)$ of X into Y due to Lemma 4.14. Note that all vertex sets $N_M[x]$ are pairwise disjoint for all $x \in X$. Due to Proposition 4.19, every s-partial vertex cover of H has to contain at least one vertex from each vertex set $N_M[x]$. Consequently, as H is a subgraph of G, every s-partial vertex cover of G has to contain at least one vertex from each vertex set $N_M[x]$ as well. Hence, $|S \cap (X \cup Y)| \geq |X|$. It follows that $|S| = |S \setminus (X \cup Y)| + |S \cap (X \cup Y)| \geq |S \setminus (X \cup Y)| + |X| = |S^*|$. Thus, $|S| \geq |S^*|$.

Lastly, we show that if $\{u, v\} \in E(G)$ is covered by S, then $\{u, v\}$ is covered by S^* as well. Let $\{u, v\} \in E(G)$ be covered by S. If $u \notin Y$ or $v \notin Y$, then we observe that $\{u, v\}$ is covered by S^*. Furthermore, it cannot happen that u and v are both inside Y because of the first assumption of Lemma 4.20.

All in all, S^* is a minimum-size s-partial vertex cover of G such that $S^* \cap (X \cup Y) = X$. Thus, Lemma 4.20 holds. □

Lemma 4.20 will help us to construct a data reduction rule for Partial Vertex Cover later on. Before we present it, we show how to find a bipartite subgraph H of G which fulfills the requirements to apply Lemma 4.20 to it.

Complying with the Expansion Lemma

In the following, we show for a graph G and a half-integral optimal solution of LPVC(G) that if $V_0(G)$ is large enough, then the expansion lemma is applicable to the bipartite graph H with vertex bipartition $(V_A(H) := V_1(G), V_B(H) := V_0(G))$. In the end, this allows us to find an upper bound on the size of $V_0(G)$.

First, we formally define the bipartite subgraph H of G which we will use to apply Lemma 4.20 later on.

Definition 4.21. Let G be a graph and $(x_v)_{v \in V(G)}$ be a half-integral optimal solution of LPVC(G). Then, $H_{G,(x_v)_{v \in V(G)}} \subseteq G$ is the bipartite subgraph of G such that $V_A(H_{G,(x_v)_{v \in V(G)}}) := V_1(G)$, $V_B(H_{G,(x_v)_{v \in V(G)}}) := V_0(G)$, and $E(H_{G,(x_v)_{v \in V(G)}}) := E(G, V_1(G), V_0(G))$.

If G and $(x_v)_{v \in V(G)}$ are clear from the context, then we omit their subscripts. Next, we show that if $V_0(G)$ large enough, then Lemma 4.14 is applicable to H.

Proposition 4.22. *Let G be a graph without isolated vertices, $(x_v)_{v \in V(G)}$ be a half-integral optimal solution of* LPVC*(G), H be the bipartite subgraph of G as in Definition 4.21, and $s \in \mathbb{N}$. If $|V_0(G)| \geq (s+1) \cdot |V_1(G)|$, then there exists a minimum-size s-partial vertex cover S of G such that $S \cap (X \cup Y) = X$, with X and Y being the vertex sets as specified in Lemma 4.14 when applied to H and $q := s + 1$.*

Proof. We show that H and s satisfy both requirements to apply Lemma 4.20.

First, the neighborhood of $V_B(H) = V_0(G)$ is part of $V_A(H) = V_1(G)$ due to Observation 4.11.

Next, we show that H satisfies all requirements to apply Lemma 4.14 with q to it. It holds that H is a bipartite graph and $q \in \mathbb{N}_+$. Furthermore, by our assumption, $|V_B(H)| = |V_0(G)| \geq q \cdot |V_1(G)| = q \cdot |V_A(H)|$, and so the first requirement of Lemma 4.14 holds. Hence, it remains to show that $V_B(H) = V_0(G)$ does not contain isolated vertices with respect to H. Due to Observation 4.11, $N_G(V_B(H)) = N_G(V_0(G)) \subseteq V_1(G) = V_A(H)$ holds. Hence, if there would be an isolated vertex v in $V_B(H)$ with respect to H, then v would be isolated in G as well. However, as we assume that G does not contain any isolated vertices, there are no isolated vertices in $V_B(H)$ with respect to H. Hence, the second requirement of Lemma 4.14 is fulfilled as well.

Thus, Lemma 4.20 is applicable to G with respect to H and s, so Proposition 4.22 holds. □

Next, we put everything together to find an upper bound on the size of $V_0(G)$.

An Upper Bound on $|V_0(G)|$

To find an upper bound on the size of $V_0(G)$ for our problem kernel, we first present a data reduction rule which is based on Proposition 4.22. Afterwards, we prove that this data reduction rule together with Reduction Rule 4.3.1 leads to $(s+1) \cdot (\kappa + s)$ being an upper bound on the size of $V_0(G)$.

Reduction Rule 4.3.2. *Let (G, κ, s) be an instance of* Partial Vertex Cover *with κ being the solution size and s the maximum number of edges not covered. Furthermore, let $(x_v)_{v \in V(G)}$ be a half-integral, optimal solution of* LPVC*(G).*

First, exhaustively apply Reduction Rule 4.2.1 to (G, κ, s) to remove all isolated vertices from G. Next, let H be the bipartite subgraph of G as in Definition 4.21.

In the case of $|V_0(G)| \geq (s+1) \cdot |V_1(G)|$, let X and Y be resulting vertex sets as specified in Lemma 4.14 from applying Lemma 4.14 to H and $q := s+1$. If $\kappa' := \kappa - |X| < 0$, then return a trivial `NO`-instance of Partial Vertex Cover*, otherwise return the* Partial Vertex Cover *instance $(G' := G - (X \cup Y), \kappa' := \kappa - |X|, s' := s)$ together with a half-integral optimal solution $(x'_v)_{v \in V(G')} := (x_v)_{v \in V(G')}$ of* LPVC*(G').*

Lemma 4.23. *Reduction Rule 4.3.2 is safe and it is exhaustively applicable in $\mathcal{O}(\kappa \cdot (n+m)^2)$ time.*

Proof. We prove the correctness of Reduction Rule 4.3.2, that is we show (G, κ, s) is a `YES`-instance of Partial Vertex Cover if and only if (G', κ', s') is a `YES`-instance of Partial Vertex Cover. We assume that Reduction Rule 4.3.2 is applied, as otherwise the correctness holds trivially. Thus, we may assume for both directions that $|V_0(G)| \geq (s+1) \cdot |V_1(G)|$, as otherwise Reduction Rule 4.3.2 was not applied.

"$\Rightarrow$": First, G does not contain any isolated vertices because we already exhaustively applied Reduction Rule 4.2.1. Thus, we may assume that there exists a minimum-size s-partial vertex cover S of G such that $S \cap (X \cup Y) = X$ due to Proposition 4.22. As (G, κ, s) is a `YES`-instance of Partial Vertex Cover, we conclude that $|S| \leq \kappa$. Hence, $S' := S \setminus X$ is an s-partial vertex cover of G', because all edges incident to X in G are not part of G'. Next, S' has size $|S| - |X| \leq \kappa - |X| = \kappa'$. As κ' is non-negative because (G, κ, s) is a `YES`-instance of Partial Vertex Cover, Reduction Rule 4.3.2 does not return a trivial `NO`-instance of Partial Vertex Cover. Additionally, $(x'_v)_{v \in V(G')}$ is an optimal solution of LPVC(G'), as otherwise $(x_v)_{v \in V(G)}$ would not have been an optimal solution of LPVC(G). All in all, the returned instance (G', κ', s') is a `YES`-instance of Partial Vertex Cover, because S' is an s-partial vertex cover of size at most κ' of G'.

"$\Leftarrow$": Let S' be an s'-partial vertex cover of G' of size at most κ'. We prove that $S := S' \cup X$ is an s-partial vertex cover of G. Because $N_G(Y) \subseteq X$ holds due to Observation 4.11 and Lemma 4.14, all edges in $E(G) \setminus E(G')$ are incident to some vertex in X. Hence, S covers in G all edges which are not part of G', and all except at most $s' = s$ edges of G'. Thus, S is an s-partial vertex cover of G of size $|S'| + |X| \leq \kappa' + |X| = \kappa$. All in all, (G, κ, s) is a `YES`-instance of Partial Vertex Cover.

To apply Reduction Rule 4.3.2 once, we first exhaustively apply Reduction Rule 4.2.1 in $\mathcal{O}(n_G + m_G)$ time. Next, we compute H in $\mathcal{O}(n_G + m_G)$ time. Lastly, we apply Lemma 4.14 in $\mathcal{O}((n_G + m_G)^2)$ time. Thus, it takes $\mathcal{O}((n_G + m_G)^2)$ time to apply Reduction Rule 4.3.2 once.

Because the vertex set X as specified in Lemma 4.14 is non-empty, $\kappa' := \kappa - |X|$ is strictly smaller than κ. As we return a trivial `NO`-instance of Partial Vertex Cover in the case of $\kappa' < 0$, we have to apply Reduction Rule 4.3.2 at most $\kappa + 1$ times. Thus, it takes $\mathcal{O}(\kappa \cdot (n_G + m_G)^2)$ time to exhaustively apply Reduction Rule 4.3.2 to a Partial Vertex Cover instance. □

Next, we show that after exhaustively applying Reduction Rule 4.3.2, we find an upper bound on the number of vertices in $V_0(G)$.

Lemma 4.24. *Let (G, κ, s) be an instance of* Partial Vertex Cover *and $(x_v)_{v\in V(G)}$ be a half-integral optimal solution of* LPVC*(G) such that Reduction Rule 4.3.2 is not applicable anymore. Then, $|V_0(G)| < (s+1) \cdot |V_1(G)|$.*

Proof. To start, we assume that Reduction Rule 4.3.2 has not returned a trivial `NO`-instance of Partial Vertex Cover, as otherwise Lemma 4.24 holds trivially.

Next, as we cannot apply Reduction Rule 4.3.2 to (G, κ, s) and $(x_v)_{v\in V(G)}$, we observe that at least one of the following three claims has to hold:

1. $\kappa < 0$, or
2. there is an isolated vertex in G, or
3. $|V_0(G)| < (s+1) \cdot |V_1(G)|$.

If all three claims would not hold, then we could apply Reduction Rule 4.3.2 again, which would lead to a contradiction.

If the first claim would hold, then Reduction Rule 4.3.2 would have returned a trivial `NO`-instance of Partial Vertex Cover, which we assume to not be the case. The second claim cannot hold as well, as Reduction Rule 4.3.2 removes all isolated vertices by exhaustively applying Reduction Rule 4.2.1 in each iteration. Thus, the third claim $|V_0(G)| < (s+1) \cdot |V_1(G)|$ has to hold, which means that Lemma 4.24 holds. □

Next, we put everything together to compute our problem kernel. We will also show how to find a smaller upper bound on the total number of vertices of this problem kernel by studying the relations between the sizes of $V_0(G), V_{1/2}(G)$, and $V_1(G)$ in more detail.

4.3.4 Analysis of Our Kernel

We will discuss how to efficiently compute our problem kernel for Partial Vertex Cover and show that it contains strictly less than $(s+2) \cdot (\kappa + s)$ vertices, where κ is the solution size and s is the maximum number of edges not covered. Note that if $s = 0$, then we obtain a 2κ-vertex problem kernel for Vertex Cover. Additionally, for all $\kappa \in \mathbb{N}$, $s \in \mathbb{N}_+$, we will provide a graph with exactly $(s+2)\cdot(\kappa+s)-1$ vertices for which we cannot apply any of our data reduction rules. This means that we need new data reduction rules to find a problem kernel with less vertices for Partial Vertex Cover with respect to the combined parameter $\kappa + s$.

We compute our problem kernel for a Partial Vertex Cover instance (G, κ, s) in the following way. First, we compute a half-integral optimal solution for LPVC(G) in $\mathcal{O}(m\sqrt{n})$ time as discussed in Section 4.3.1, and afterwards we apply Reduction Rule 4.3.1 once in linear time. Finally we exhaustively apply Reduction Rule 4.3.2 in $\mathcal{O}(\kappa \cdot (n+m)^2)$ time. Note that none of our data reduction rule increases κ or s.

We will show that $(s+2) \cdot (\kappa + s)$ is an upper bound on the number of vertices by combining the following inequalities.

$$|V_1(G)| + 1/2 \cdot |V_{1/2}(G)| \leq \kappa + s \tag{4.3}$$
$$|V_0(G)| < (s+1) \cdot |V_1(G)| \tag{4.4}$$

Both inequalities have to hold after exhaustively applying Reduction Rules 4.3.1 and 4.3.2, see Lemmas 4.18 and 4.24. Next, we combine both inequalities to find an upper bound on the number of vertices in G.

$$\begin{aligned}
|V(G)| &= |V_0(G)| + |V_{1/2}(G)| + |V_1(G)| \\
&\overset{\text{Inequality 4.4}}{<} (s+2) \cdot |V_1(G)| + |V_{1/2}(G)| \\
&= s \cdot |V_1(G)| + 2 \cdot (|V_1(G)| + 1/2 \cdot |V_{1/2}(G)|) \\
&\overset{\text{Inequality 4.3}}{\leq} s \cdot |V_1(G)| + 2 \cdot (\kappa + s) \\
&\overset{\text{Inequality 4.3}}{\leq} (s+2) \cdot (\kappa + s)
\end{aligned} \tag{4.5}$$

Hence, $(s+2) \cdot (\kappa + s) - 1$ is an upper bound on the number of the vertices in G.

As we compute our problem kernel in $\mathcal{O}(m\sqrt{n} + \kappa \cdot (n+m)^2) = \mathcal{O}(\kappa \cdot (n+m)^2)$ time and have strictly less than $(s+2) \cdot (\kappa+s)$ vertices in the resulting graph, we have proven Theorem 4.9.

Theorem 4.9. Partial Vertex Cover *admits a problem kernel with strictly less than $(s+2) \cdot (\kappa+s)$ vertices, where κ is the solution size and s is the maximum number of edges not covered. This problem kernel is computable in $\mathcal{O}(\kappa \cdot (n+m)^2)$ time.*

Note that for the special case $s = 0$, Partial Vertex Cover coincides with the Vertex Cover problem. Thus, we obtain a 2κ-vertex problem kernel for Vertex Cover, therefore matching the size of the problem kernel of Theorem 4.12.

For constant s, Theorem 4.9 admits a problem kernel for Partial Vertex Cover with $\mathcal{O}(\kappa)$ vertices, which is computable in $\mathcal{O}(\kappa \cdot (n+m)^2)$ time. This stands in contrast to Theorem 4.8, where we presented a problem kernel for Partial Vertex Cover that for constant s admits a problem kernel with $\mathcal{O}(\kappa^2)$ vertices, which is computable in linear time.

Because of the duality of s-Defective Clique and Partial Vertex Cover as presented in Observation 4.1, we conclude from Theorem 4.9 that there exists a similar linear-vertex problem kernel for s-Defective Clique with respect to the combined parameter $(n - k) + s$, where k is the solution size and s the maximum number of missing edges.

Additionally, we show that our analysis for Theorem 4.9 is tight by constructing for each $\kappa \in \mathbb{N}, s \in \mathbb{N}_+$ a graph $G_{\kappa,s}$ which matches the upper bound on the number of vertices from Inequality 4.5. Thus, we need new data reduction rules to find a smaller upper bound on the number of vertices.

Proposition 4.25. *For all $\kappa \in \mathbb{N}, s \in \mathbb{N}_+$, there exists a graph $G_{\kappa,s}$ with exactly $(s+2) \cdot (\kappa+s) - 1$ vertices and a half-integral optimal solution $(y_v)_{v \in V(G_{\kappa,s})}$ for* LPVC*$(G_{\kappa,s})$ for which neither Reduction Rule 4.3.1 nor Reduction Rule 4.2.1 nor Reduction Rule 4.3.2 can be applied.*

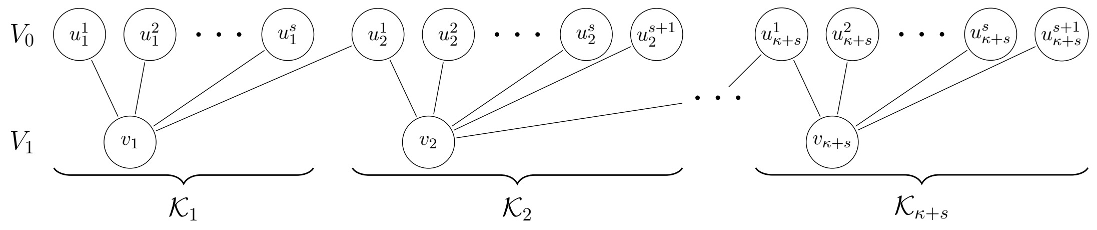


Figure 4.3: An illustration of the graph $G_{\kappa,s}$ for some $\kappa \in \mathbb{N}, s \in \mathbb{N}_+$ and a half-integral optimal solution $(y_v)_{v\in V(G_{\kappa,s})}$ for LPVC$(G_{\kappa,s})$. The graph $G_{\kappa,s}$ is build upon the stars $\mathcal{K}_1,\dots,\mathcal{K}_{\kappa+s}$. Each graph $G_{\kappa,s}$ contains $(s+2)\cdot(\kappa+s)-1$ vertices and we can neither apply Reduction Rule 4.3.1 nor Reduction Rule 4.2.1 nor Reduction Rule 4.3.2 to the Partial Vertex Cover instance $(G_{\kappa,s},\kappa,s)$ and $(y_v)_{v\in V(G_{\kappa,s})}$.

Proof. We build $G_{\kappa,s}$ out of several stars. For any κ, s, each graph $G_{\kappa,s}$ consists of the disjoint stars $\mathcal{K}_1,\mathcal{K}_2,\dots,\mathcal{K}_{\kappa+s}$, where $\mathcal{K}_1$ is a star $K_{1,s}$ with v_1 as its center and s leaves $u_1^1,u_1^2,\dots,u_1^s$, while $\mathcal{K}_i$ is a star $K_{1,s+1}$ with v_i as its center and $s+1$ leaves $u_i^1,u_i^2,\dots,u_i^s,u_i^{s+1}$ for $i\geq 2$. Additionally, for the first $\kappa+s-1$ stars, we add an edge between the center v_i of the i-th star and the first leaf of the next star u_{i+1}^1. Formally, we define $V(G_{\kappa,s}) := \bigcup_{i\in[\kappa+s]} V(\mathcal{K}_i)$, and $E(G_{\kappa,s}) := (\bigcup_{i\in[\kappa+s]} E(\mathcal{K}_i)) \cup \{\{v_i,u_{i+1}^1\} \mid i \in [\kappa+s-1]\}$. See Figure 4.3 for an illustration of $G_{\kappa,s}$.

Next, we discuss why we cannot apply any of our data reduction rules to $G_{\kappa,s}$ with $\kappa \in \mathbb{N}, s\in\mathbb{N}_+$. We define a half-integral optimal solution $(y_v)_{v\in V(G_{\kappa,s})}$ for LPVC$(G_{\kappa,s})$, which we would need to apply Reduction Rule 4.3.1 or Reduction Rule 4.3.2. In general, we observe that for a star G with at least one leaf, a half-integral optimal solution for LPVC(G) is constructable by assigning 1 to the corresponding variable of its center, and 0 to the variables of all leaves. Thus, for any κ, s, the list of variables $(y_v)_{v\in V(G_{\kappa,s})}$ with $y_{v_i} := 1$, $y_{u_1^j} := 0$, and $y_{u_i^{j'}} := 0$, where $i \in [\kappa+s], j\in[s], j'\in[s+1]$, is a half-integral optimal solution for LPVC$(G_{\kappa,s})$. This is because our additional edges between the stars do not decrease the optimal value of LPVC$(G_{\kappa,s})$, and $(y_v)_{v\in V(G_{\kappa,s})}$ is still a feasible solution of LPVC$(G_{\kappa,s})$. Note that this argument would not apply when we allowed s to be zero. Hence, $V_1(G_{\kappa,s})$ contains $\kappa+s$ vertices, $V_0(G_{\kappa,s})$ contains $(s+1)\cdot(\kappa+s)-1$ vertices, and $V_{1/2}(G_{\kappa,s})$ contains no vertices. Overall, $G_{\kappa,s}$ contains $(s+2)\cdot(\kappa+s)-1$ vertices.

We observe that $G_{\kappa,s}$ does not contain any isolated vertices, so we cannot apply Reduction Rule 4.2.1. For $(y_v)_{v\in V(G_{\kappa,s})}$, the inequality $|V_1(G_{\kappa,s})| + 1/2\cdot\left|V_{1/2}(G_{\kappa,s})\right| \leq \kappa+s$ holds, so we cannot apply Reduction Rule 4.3.1 to $G_{\kappa,s}$ and $(y_v)_{v\in V(G_{\kappa,s})}$ either. Lastly, the inequality $|V_0(G_{\kappa,s})| \geq (s+1)\cdot|V_1(G_{\kappa,s})|$ does not hold, so we cannot apply Reduction Rule 4.3.2 either. Thus, we cannot apply any of our data reduction rules to $G_{\kappa,s}$ with respect to $(y_v)_{v\in V(G_{\kappa,s})}$. □

Due to Proposition 4.25, our analysis is tight. Hence, we need new data reduction rules to find a problem kernel for Partial Vertex Cover with respect to the combined parameter $\kappa+s$ with at most $(s+2-\varepsilon)\cdot(\kappa+s)$ vertices for any $\varepsilon>0$.

Finally, we show how the results of this chapter link to the rest of this thesis, and how they relate to other work. In the context of this thesis, we presented that the three

Clique relaxations s-Club, s-Plex, and s-Defective Clique are fixed-parameter tractable with respect to the combined parameter "large gap" plus "respective relaxation parameter" $(n-k)+s$. Furthermore, with Proposition 4.3, we also showed that we can decide all three problems by "branching over a set of vertices which cannot be part of the same solution". In Chapter 5, we will find the "essence" of these three branching algorithms and formalize it. This formalization will be one of our two ingredients towards a more general framework for deciding Clique relaxations in Chapter 5. Additionally, we will also show how to apply our two problem kernels from Theorems 4.8 and 4.9 for Partial Vertex Cover to s-Defective Clique in Chapter 6.

We briefly mention that for the Bounded-Degree Vertex Deletion problem, which is strongly related to Partial Vertex Cover, a similar "Buss-like" problem kernel like Theorem 4.8 is known [MNS12]. However, for a constant "relaxation parameter", we only know that there exists a quasi-linear–vertex problem kernel for Bounded-Degree Vertex Deletion with respect to the solution size [Fel+11], which stands in contrast to our linear-vertex problem kernel from Theorem 4.9 for Partial Vertex Cover with respect to the solution size.

Chapter 5

A Smaller Gap and a Gap-Framework

In Chapter 3, we have shown that we cannot easily re-formulate the KWB-algorithm of Komusiewicz [Kom11, Proposition 5.4] (re-introduced and implemented by Walteros and Buchanan [WB20, Theorem 1]) for MAXIMUM CLIQUE to clique relaxations with respect to the clique-core gap $g := d+1-\omega$, where d is the degeneracy of G, and ω is the order of a maximum clique in G. This is because s-DEFECTIVE CLIQUE, s-PLEX, and s-CLUB are $W[1]$- or para-NP-hard with respect to the clique-core gap. Furthermore, in Chapter 4, we have presented that these three CLIQUE relaxations are fixed-parameter tractable with respect to the combined parameter "large gap" plus "respective relaxation parameter" $(n-k)+s$, where $k \in \mathbb{N}$ is the solution size.

Unfortunately, the maximum-order of a relaxed clique in real-world graphs is relatively small (see Tables A.5 to A.8), therefore the parameter $n-k$ is often almost as large as n itself. The goal of this chapter is to find a "new gap" which is smaller than $n-k$ in real-world graphs, but still can be used to develop FPT-algorithms for CLIQUE relaxations with respect to this "new gap". In other words, as in practice we often want to find a solution of *maximum order*, our goal is to find a better upper bound than n on the maximum order of a relaxed clique, and provide a general "gap-framework" for solving MAXIMUM CLIQUE relaxations. Next, we will study the main ingredients of the KWB-algorithm in more detail and generalize these ingredients in a way so that they are applicable to MAXIMUM CLIQUE relaxations.

The KWB-algorithm has the following two ingredients:

1. Constructing a Turing kernel, that is splitting the input graph into many subgraphs with at most $d+1$ vertices, where all subgraphs can be solved separately, and

2. solving each subgraph by branching over two vertices which cannot be part of the same solution, that is they are non-adjacent, until the resulting instance induces a clique.

Recall from Komusiewicz [Kom16, Theorem 5.4] that each subgraph in the Turing kernel has at most $d+1$ vertices and there is an algorithm for finding a maximum clique in $\mathcal{O}^*(2^{n-\omega})$ time, thus we can combine both ideas to find a maximum clique in $\mathcal{O}^*(2^{d+1-\omega})$ time. To understand how a new "relaxed gap" can be defined for clique relaxations, we will present next the two ingredients of the KWB-algorithm again in more detail.

Turing kernelization. It is known that a clique $S \subseteq V(G)$ in a graph G has diameter exactly one. Hence, for each $v \in S$ it holds that $S \subseteq N[v]$. Thus, for the leftmost vertex v of S with respect to some degeneracy ordering, the closed right-neighborhood of v contains S completely. This means that one can propagate the search of finding a clique in G to finding a clique in the n subgraphs induced by the closed right-neighborhoods of all vertices in G with respect to some degeneracy ordering. Note that each of these subgraphs contains at most $d+1$ vertices. This technique is known as Turing kernelization, see Section 2.1 for the precise definition. Note that this means that a clique in a graph has order at most $d+1$.

Recall that the x-neighborhood of a vertex v contains all vertices with distance at most x to v in G, except v itself. We will see in Section 5.2.1 that several Clique relaxations also implicitly have an upper bound on the diameter of their solutions. This leads to the idea of generalizing the degeneracy and degeneracy ordering of a graph by considering the x-neighborhood of the vertices, rather than the standard neighborhood. In Section 5.1, we will see that there are two natural generalizations of the standard degeneracy, both with strengths and weaknesses. To make it easier for the reader, we will refer to this concept as the *x-degeneracy* d_x and *x-degeneracy ordering* until the start of Section 5.1.

We will see in Section 5.2.1 that if there exists an upper bound $x \in \mathbb{N}$ on the diameter of the solution of a Maximum Clique relaxation, then we can propagate the search of finding a relaxed clique in G to finding a relaxed clique in the n subgraphs induced by the *closed x-right-neighborhoods* of G with respect to some x-degeneracy ordering, thus effectively constructing a Turing kernel. Note that $d_x + 1$ is an upper bound on the maximum order of such a relaxed clique. Such an upper bound on the diameter can only exist when all solutions of the Maximum Clique relaxation are connected, which is not always the case (for example, two isolated vertices induce a 1-defective clique as well as a 2-plex). We already discussed the advantages of searching only *connected* solutions in Section 1.1 and came to the conclusion that this "restriction" is well-motivated. For the rest of the thesis, we assume that G is connected.

Incompatible vertices. The second property of Maximum Clique that the KWB-algorithm uses is that one can find a clique of order $\omega \in \mathbb{N}$ in a graph G by branching at most $n - \omega$ times over removing one of two non-adjacent vertices, as both vertices cannot be part of the same clique. This idea was named branching over two *incompatible vertices* by Komusiewicz et al. [Kom+19], and in Section 5.2.2 we will recall this concept in more detail and generalize it for several Maximum Clique relaxations to branch over some constant number of incompatible vertices.

Combining both ingredients. After having discussed both ingredients of the KWB-algorithm, we observe that we can combine them to get the following framework for some clique relaxation Π: "If each graph that satisfies Π has diameter at most $x \in \mathbb{N}$, and if we can find in polynomial time a constant number of vertices which cannot be part of the same solution, then one can find a subgraph of maximum order satisfying Π in $2^{\mathcal{O}(d_x+1-\omega_\Pi)} \cdot n^{\mathcal{O}(1)}$ time." Here, ω_Π is the maximum order of a subgraph G' of G such that G' is a relaxed clique Π. We say that $(d_x + 1) - \omega_\Pi$ is a *gap*, as $d_x + 1$ is an upper bound on the order of G'. Hence, $(d_x + 1) - \omega_\Pi$ is the gap between the maximum possible order $d_x + 1$ of G' in G and the actual maximum order of such a subgraph in G. In Section 5.2, we will present this *gap-framework* in more detail.

Before we present our generalization of the KWB-algorithm, we will study next the x-degeneracy of a graph for $x \geq 2$ in more detail, as it is an important part of our Turing kernelization. The goal is to relate this new family of parameters to known parameters, as well as to study the complexity of computing the x-degeneracy of a graph.

5.1 Generalizing Degeneracy

After having motivated the study of a generalized degeneracy of a graph for $x \geq 2$, we will formally define it here. We will see that there are two natural ways to generalize the standard degeneracy concept, and we will show that both ways will lead to non-equivalent definitions. Afterwards, we demonstrate how both generalizations relate to well-known graph parameters. Furthermore, we show how to compute both generalizations in polynomial time. Finally, we prove that an algorithm computing any of the two generalizations in $\mathcal{O}(n^{1.999})$ time would break the Strong Exponential Time Hypothesis, which would result in a major breakthrough in parameterized complexity theory.

Introducing generalizations. It is known that the standard degeneracy of a graph G can be defined in two equivalent ways. We will generalize both options and show that they are non-equivalent. Recall that for $x \in \mathbb{N}_+$, $\delta_x(G)$ is the minimum size of an x-neighborhood $N_x(v)$ for some vertex v in G.

Definition 5.1 ([Pic15; Tru+13])**.** Let G be a graph and $x \in \mathbb{N}$. The *weak x-degeneracy* of G is the minimum number $\alpha_x(G) \in \mathbb{N}$ such that for all $G' \subseteq G$ it holds that $\delta_x(G') \leq \alpha_x(G)$.

Furthermore, we say that $\preceq_{\alpha_x} := (v_1, v_2, \ldots, v_n)$ is a *weak x-degeneracy ordering* of G if for all $i \in [n]$ it holds that $\delta_x(G_i) = |N_{x,G_i}(v_i)|$, where $G_i := G[v_i, v_{i+1}, \ldots, v_n]$. Additionally, $\mathrm{wrN}_x(v_i, G, \preceq_{\alpha_x}) := N_{x,G_i}(v_i)$ is the (open) *weak x-right-neighborhood* of a vertex $v_i \in V(G)$ with respect to $\preceq_{\alpha_x}$, and $\mathrm{wrN}_x[v_i, G, \preceq_{\alpha_x}] := \mathrm{wrN}_x(v_i, G, \preceq_{\alpha_x}) \cup \{v_i\}$ is the *closed weak x-right-neighborhood* of v_i with respect to $\preceq_{\alpha_x}$. If G or $\preceq_{\alpha_x}$ are clear from the context, then we will omit their arguments. See Figure 5.1 for an example of a weak 2-degeneracy ordering, which we will discuss later in more detail when comparing both generalizations of the standard degeneracy concept.

We observe that G has weak x-degeneracy α_x if and only if in a weak x-degeneracy ordering of G each vertex has at most α_x x-right-neighbors due to an argument of Matula and Beck [MB83, Chapter 2]. For the sake of completeness, we provide the whole proof.

Observation 5.2. *Let G a graph and $x \in \mathbb{N}$. Furthermore, let $\preceq_{\alpha_x} := (v_1, v_2, \ldots, v_n)$ be a weak x-degeneracy ordering of G, and let α'_x be the maximum size of a weak x-right-neighborhood with respect to $\preceq_{\alpha_x}$. Then, $\alpha_x = \alpha'_x$.*

Proof. "$\alpha_x \geq \alpha'_x$": Let $i \in [n]$ be arbitrary but fixed, and $G_i := G[v_i, v_{i+1}, \ldots, v_n]$. As v_i has the minimum x-degree of G_i, and by Definition 5.1 it holds that $\delta_x(G_i) \leq \alpha_x$, we conclude that $\mathrm{wrN}_x(v_i)$ has size at most α_x.

"$\alpha_x \leq \alpha'_x$": Let $G' \subseteq G$ be an arbitrary subgraph of G. Furthermore, let v be the leftmost vertex of G' with respect to $\preceq_{\alpha_x}$. As $\mathrm{wrN}_x(v)$ has size at most α'_x, we conclude that the minimum x-degree of G' is at most α'_x. □

Next, we generalize the second way to define the standard degeneracy.

Definition 5.3. Let G be a graph and $x \in \mathbb{N}$. The *strong x-degeneracy* of G is the minimum number $\beta_x \in \mathbb{N}$ such that for the vertices of G there exists a linear ordering $\preceq_{\beta_x}$ such that no vertex $v \in V(G)$ has more than β_x of its x-neighbors right to it with respect to $\preceq_{\beta_x}$, formally,

$$\beta_x(G) := \min_{\preceq \text{ a linear ordering of } V(G)} \{ \max_{v \in V(G)} \{ |\{u \in V(G) \setminus \{v\} \mid v \preceq u; \mathrm{dist}(u, v) \leq x\}| \} \}.$$

Additionally, a corresponding linear ordering $\preceq_{\beta_x}$ is a *strong x-degeneracy ordering* of G. Furthermore, $\mathrm{srN}_x(v_i, G, \preceq_{\beta_x}) := N_{x,G}(v_i) \cap \{v_{i+1}, v_{i+2}, \ldots, v_n\}$ is the (open) *strong x-right-neighborhood* of a vertex $v_i \in V(G)$ with respect to $\preceq_{\beta_x}$, and $\mathrm{srN}_x[v_i, G, \preceq_{\beta_x}] := \mathrm{srN}_x(v_i, G, \preceq_{\beta_x}) \cup \{v_i\}$ is the *closed strong x-right-neighborhood* of v_i with respect to $\preceq_{\beta_x}$. If G or $\preceq_{\beta_x}$ are clear from the context, then we will omit their arguments. See Figure 5.1 for an example of a strong 2-degeneracy ordering, which we discuss later in more detail when comparing the weak x-degeneracy and strong x-degeneracy.

We observe that the strong x-degeneracy of a graph is the same as the standard degeneracy of its power graph G^x, where the power graph G^x is the graph where two distinct vertices are adjacent if and only if they have distance at most x.

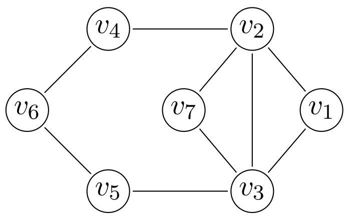
Observation 5.4. *Let G be a graph, $x \in \mathbb{N}_+$, and G^x be the power graph of G. Then, $\beta_x(G) = d(G^x)$.*

Proof. Let G be a graph and $\preceq$ be some linear ordering of the vertices of G. To prove Observation 5.4 we will show that $\mathrm{srN}_x(v, G, \preceq) = \mathrm{rN}(v, G^x, \preceq)$ for all $v \in V(G)$. This is sufficient, as we define the strong x-degeneracy of a graph as the maximum size of a strong x-right-neighborhood, and the standard degeneracy as the maximum size of a right-neighborhood with respect to a linear ordering.

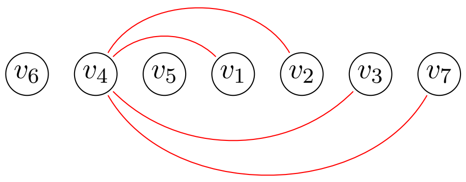
"$u \in \mathrm{srN}_x(v, G, \preceq) \Rightarrow u \in \mathrm{rN}(v, G^x, \preceq)$": Due to the definition of the strong x-right-neighborhood, v and u have distance at most x in G. Hence, v and u are adjacent in G^x. Furthermore, u is right of v with respect to $\preceq$. Finally, this means that $u \in \mathrm{rN}(v, G^x, \preceq)$.

"$u \in \mathrm{srN}_x(v, G, \preceq) \Leftarrow u \in \mathrm{rN}(v, G^x, \preceq)$": As v and u are adjacent in G^x, v and u have distance at most x in G. Hence, u is in the x-neighborhood of v, which means that $u \in \mathrm{srN}_x(v, G, \preceq)$. □

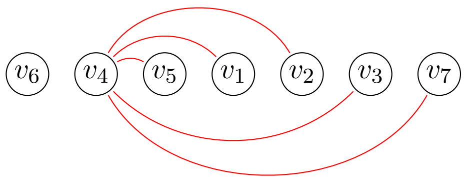
Relation between the generalizations. To explain the difference between the weak x-right-neighborhood and the strong x-right-neighborhood on a high level, let G be a graph, $(v_1, v_2, \ldots, v_n)$ be a linear ordering of $V(G)$, $i \in [n]$, and $G' := G[v_i, v_{i+1}, \ldots, v_n]$. While v_i has distance at most x to all vertices in its weak x-right-neighborhood with respect to G', it can happen that the distance between v_i and a vertex in its strong



(a) An illustration of the graph G from Bourjolly, Laporte, and Pesant [BLP02, Figure 4].



(b) G rearranged with respect to a weak 2-degeneracy ordering of G. Red "edges" illustrate the weak 2-right-neighborhood of v_4.



(c) G rearranged with respect to a strong 2-degeneracy ordering of G. Red "edges" illustrate the strong 2-right-neighborhood of v_4.

Figure 5.1: An illustration of a graph G for which the weak 2-degeneracy is strictly smaller than its strong 2-degeneracy. We observe that $\mathrm{srN}_2(v_4) \setminus \mathrm{wrN}_2(v_4) = \{v_5\}$.

x-right-neighborhood is greater than x with respect to G'. For a concrete example, we consider for $x = 2$ the graph G as illustrated in Figure 5.1. By removing a minimum 2-degree vertex in each step, it is easy to see that the linear ordering $(v_6, v_4, v_5, v_1, v_2, v_3, v_7)$ is a weak 2-degeneracy ordering of G. Furthermore, the same linear ordering is also a strong 2-degeneracy ordering of G, as v_6 is the vertex of minimum degree in G^2, and $G^2 - v_6$ is a clique. Now we observe that $\mathrm{wrN}_2(v_4)$ does not contain v_5, as the distance of v_4 and v_5 in $G - v_6$ is three. However, $\mathrm{srN}_2(v_4)$ contains v_5 (although they have distance three in $G - v_6$), as v_5 is in the 2-neighborhood of v_4 with respect to G and it is right of v_5 with respect to the linear ordering.

Now, we study the relation of the weak x-degeneracy and strong x-degeneracy of a graph in more detail. Let G be a graph and $\preceq$ be some linear ordering of the vertices of G. It is easy to see that the weak x-right-neighborhood of a vertex v is a subset of the strong x-right-neighborhood of v. Hence, the weak x-degeneracy of a graph is at most as large as its strong x-degeneracy.

Next, we show that the strong x-degeneracy of a graph can be strictly larger than the weak x-degeneracy, hence Definitions 5.1 and 5.3 are non-equivalent. Let G be the graph as illustrated in Figure 5.1. Recall that the linear ordering $(v_6, v_4, v_5, v_1, v_2, v_3, v_7)$ is both a weak x-degeneracy ordering and a strong x-degeneracy ordering of G. We observe that for this linear ordering, the maximum size of a weak x-right-neighborhood is four, while the size of $\mathrm{srN}_2(v_4)$ is five. Thus, $\alpha_x(G) < \beta_x(G)$.

As the weak x-degeneracy of a graph is at most its strong x-degeneracy, we will focus on the weak x-degeneracy when constructing our Turing kernel in Section 5.2.1.

5.1.1 Relation to Other Parameters

We will show that the weak and strong x-degeneracies of a graph are both strongly related to the maximum degree Δ of the graph. Recall that the weak x-degeneracy of a graph is at most its strong x-degeneracy for all $x \in \mathbb{N}_+$. Furthermore, recall that the x-neighborhood $N_x(v)$ of a vertex v in a graph G contains all vertices with distance at most x to v in G, except v itself.

Proposition 5.5. *Let $x \geq 2$ and G be a graph. Then, $\Delta \leq \alpha_x \leq \beta_x \leq \beta_{x+1} \leq \Delta^{x+2}$, where α_x is the weak x-degeneracy of G, and β_x is the strong x-degeneracy of G.*

Proof. As a warm-up, it is easy to see that for a given graph G, $\alpha_x \leq \alpha_{x+1}$ holds for all $x \in \mathbb{N}$, as $N_{x,G'}(v) \subseteq N_{x+1,G'}(v)$ holds for all $G' \subseteq G$ and $v \in V(G)$. For the same reason, $\beta_x \leq \beta_{x+1}$ holds as well.

Now, we show that the weak 2-degeneracy is an upper bound on the maximum degree of a graph. Let v be a vertex of maximum degree Δ in G, and let $G' := G[N[v]]$. As each vertex from G' contains all other vertices from G' in its 2-neighborhood, $\delta_2(G') = \Delta$. Hence, the weak 2-degeneracy of G is at least Δ.

Next, we show that Δ^{x+1} is an upper bound on the strong x-degeneracy of a graph for constant $x \in \mathbb{N}_+$. As each vertex in a graph has at most Δ neighbors, the x-neighborhood of any vertex has size at most $\sum_{i \in [x]} \Delta^x \leq \Delta^{x+1}$. Hence, in every arbitrary linear ordering $\preceq$ of the vertices, the strong x-right-neighborhood of any vertex has size at most Δ^{x+1}. $\square$

Another relation between the maximum degree and the weak x-degeneracy of a graph is that for all $x \geq 2$, $\Delta^x \leq \alpha_{x+1}$ does *not* hold for all graphs, as the weak $(x+1)$-degeneracy of a star is always $n-1$, while $\Delta^x = (n-1)^x$.

Due to Proposition 5.5, many graph parameters relate to the weak and strong x-degeneracies in the same way as to the maximum degree of a graph. Although not relevant for following the content of this thesis, we refer to Sorge and Weller [SW19] for a discussion on the relation to other parameters.

Although we can upper bound the weak and strong x-degeneracies of a graph in some polynomial of Δ, it is still of interest to study the weak and strong x-degeneracies of a graph, as they are often much smaller than Δ^x. In Table 5.1, we provide an overview of the values of the maximum degree Δ, the weak and strong 2-degeneracies α_2, β_2, and Δ^2 for ten real-world graphs. We provide more values for more real-world graphs in Tables A.1 to A.4, and explain our implementation for computing these values in Appendix A.

5.1.2 Computation of Generalized Degeneracies

Next, it is interesting how we can compute a weak x-degeneracy ordering and a strong x-degeneracy ordering of a graph G for $x \geq 2$, and how these algorithms differ from the algorithm for computing a standard degeneracy ordering. We will present an algorithm for computing the weak x-degeneracy ordering of a graph G in $\mathcal{O}(\alpha_x nm)$ time, and a strong x-degeneracy ordering of G in $\mathcal{O}(nm)$ time for any $x \in \mathbb{N}_+$. We start with the weak x-degeneracy.

Table 5.1: Values of weak/strong 2-degeneracies in some freely-available real-world graphs [DIM12; DIM93; HKN15b; LK14]. We use the following notation: n–number of vertices, m–number of edges, Δ–maximum degree, α_2–weak 2-degeneracy, β_2–strong 2-degeneracy. See Appendix A for more details.

Graph	n	m	Δ	α_2	β_2	Δ^2
`johnson8-2-4`	28	210	15	27	27	225
`jazz`	198	2,742	100	109	109	10,000
`san400_0.7_3`	400	55,860	307	399	399	94,249
`DSJC500.5`	500	62,624	286	499	499	81,796
`hamming10-4`	1,024	$4.3 \cdot 10^5$	848	1023	1023	$7.2 \cdot 10^5$
`hep-th`	8,361	15,751	50	50	50	2,500
`cond-mat-2005`	40,421	$1.8 \cdot 10^5$	278	278	278	77,284
`G_n_pin_pout`	$1 \cdot 10^5$	$5 \cdot 10^5$	25	64	71	625
`coAuthorsCiteseer`	$2.3 \cdot 10^5$	$8.1 \cdot 10^5$	1,372	1372	1372	$1.9 \cdot 10^6$
`graph_thres_01`	$7.2 \cdot 10^5$	$2.5 \cdot 10^6$	804	804	804	$6.5 \cdot 10^5$

Lemma 5.6. *Let G be a graph and $x \in \mathbb{N}_+$. Constructing a weak x-degeneracy ordering of G and computing the weak x-degeneracy α_x of G can be done in $\mathcal{O}(\alpha_x nm)$ time.*

Proof. To compute a weak x-degeneracy ordering of G, we modify the straightforward approach: Let $G' := G$ and $\preceq := ()$. While G' is non-empty, find a vertex v with minimum x-degree in G', add v to the right of $\preceq$, and proceed on $G' := G' - v$.

Next, we observe that the only vertices which have a different x-neighborhood in $G' - v$ in contrast to G' are the ones which have v inside their x-neighborhood. In other words, in each iteration we only have to re-compute the x-neighborhood of vertices inside the x-neighborhood of v with respect to G'. By the definition of the weak x-degeneracy, $|N_{x,G'}(v)| \leq \alpha_x(G)$, thus in each iteration we have to re-compute the x-neighborhood of at most $\alpha_x(G)$ vertices.

Computing the x-neighborhood of all vertices in the beginning can be done by starting a depth-bounded breadth-first-search from each vertex in G, which takes overall $\mathcal{O}(nm)$ time. Furthermore, $G' - v$ is computable in linear time. Updating the x-neighborhoods in one iteration can be done in $\mathcal{O}(\alpha_x m)$ time. As we have n iterations, this procedure takes overall $\mathcal{O}(nm + n \cdot ((n+m) + \alpha_x m)) = \mathcal{O}(\alpha_x nm)$ time. By keeping track of the x-degrees of the vertices, the weak x-degeneracy is computable as an aside. □

To compute a strong x-degeneracy ordering of G, we exploit the relation of the strong x-degeneracy of G and the standard degeneracy of the power graph G^x due to Observation 5.4.

Lemma 5.7. *Let G be a graph and $x \in \mathbb{N}_+$. Constructing a strong x-degeneracy ordering of G and computing the strong x-degeneracy β_x of G can be done in $\mathcal{O}(nm)$ time.*

Proof. It is easy to observe from the proof of Observation 5.4 that a degeneracy ordering of the power graph G^x is a strong x-degeneracy ordering of G. Hence, first we construct G^x and afterwards compute a degeneracy ordering of G^x by using the linear-time algorithm of Matula and Beck [MB83]. By keeping track of the degrees of the vertices, one can modify their algorithm to compute the strong x-degeneracy as an aside.

We can compute G^x by starting a depth-bounded breadth-first-search from each vertex, which can be done in overall $\mathcal{O}(n_G m_G)$ time. Moreover, applying the algorithm of Matula and Beck can be done in $\mathcal{O}(n_{G^x} + m_{G^x}) \subseteq \mathcal{O}(n_G^2)$ time. All in all, a strong x-degeneracy ordering of G is constructable in $\mathcal{O}(nm)$ time. □

Next, we will show that for arbitrary $x \in \mathbb{N}_+$, the running time of our algorithm computing the strong x-degeneracy is already optimal up to a poly-logarithmic factor in sparse graphs, unless the Strong Exponential Time Hypothesis breaks.

5.1.3 Running Time Lower Bounds

Next, we will show that there cannot exist an algorithm computing the weak or strong x-degeneracy of a graph in $\mathcal{O}(n^{1.999})$ time for arbitrary $x \in \mathbb{N}_+$, unless the Strong Exponential Time Hypothesis (SETH) breaks, which would lead to a major breakthrough in parameterized complexity theory. Note that this stands in contrast to the algorithm of Matula and Beck [MB83], which computes the standard degeneracy of a graph in linear time.

Before we present our result, we first recall the SETH as well as a lower bound on the running time of computing the diameter of a graph which builds upon the SETH. Afterwards, we will show that a fast algorithm computing the weak or strong x-degeneracy of a graph for arbitrary $x \in \mathbb{N}_+$ would break the lower bound on the running time of computing the diameter of a graph, hence breaking the SETH.

Fundamental Results from the Literature

The Strong Exponential Time Hypothesis (SETH) claims that for deciding CNF-SAT, one cannot find an algorithm which outperforms the trivial brute-force algorithm in the worst case. This claim was first made by Impagliazzo, Paturi, and Zane [IPZ01], but we present it as stated by Komusiewicz et al. [Kom+19, Section 4.2].

Hypothesis 5.8 (Strong Exponential Time Hypothesis (SETH) [IPZ01])**.** *There is no algorithm deciding* CNF-SAT *in* $(2-\varepsilon)^N \cdot (N+M)^{\mathcal{O}(1)}$ *time for any* $\varepsilon > 0$*, where* N *is the number of variables and* M *is the number of clauses of the input formula.*

Although the SETH builds upon an NP-hard problem, it can also be used to find conditional lower bounds on the running time of decision problems which are decidable in polynomial time. Recall that the diameter of a graph is at most $x \in \mathbb{N}$ if and only if the distance between each pair of vertices is at most x. One can easily deduce from the more general result of Roditty and Williams [RW13, Theorem 4] the following lower bound on the running time of computing the diameter of a graph.

Theorem 5.9 (Roditty and Williams [RW13, Theorem 4])**.** *Let* G *be a graph with diameter two or three. For* $\varepsilon > 0$*, there is no algorithm deciding whether* G *has diameter two or three in* $\mathcal{O}(n^{2-\varepsilon})$ *time unless the SETH breaks, even when* G *contains* $\mathcal{O}(n \log n)$ *edges.*

Next, we will use Theorem 5.9 to show similar lower bounds on the running time of computing the weak or strong x-degeneracy of a graph for $x \geq 2$, based on the SETH.

Hardness Results

Next, we apply Theorem 5.9 to show that for arbitrary $x \in \mathbb{N}_+$, there cannot exist an algorithm computing the weak or strong x-degeneracy of a graph in $\mathcal{O}(n^{1.999})$ time, unless the SETH breaks.

First, we relate the diameter to the weak and strong x-degeneracies of a graph. We observe that for all $x \in \mathbb{N}$, a graph G has diameter at most x if and only if the closed x-neighborhood of each vertex contains all vertices of G.

Proposition 5.10. *Let G be a graph and $x \in \mathbb{N}$. Then, G has diameter at most x if any only if the weak x-degeneracy of G is exactly $n-1$. Moreover, G has diameter at most x if any only if the strong x-degeneracy of G is exactly $n-1$.*

Proof. Recall that if G has diameter at most x, then the distance between all pairs of vertices in G is by definition at most x. Hence, G has diameter at most x if any only if G has minimum x-degree exactly $n-1$, where the minimum x-degree of G is the minimum size of an x-neighborhood in G.

As G is a subgraph of G, it follows that G has diameter at most x if and only if the weak x-degeneracy of G is exactly $n-1$, so our first claim holds.

For the second claim, note that G has diameter at most x if and only if G^x is a clique. As the standard degeneracy of a graph is $n-1$ if and only if it is a clique, the second claim follows. □

Due to this result, it is easy to see that the diameter of a graph is at most two if and only if its weak and strong 2-degeneracies are exactly $n-1$. Combining these results with Theorem 5.9 leads to the following result:

Theorem 5.11. *There is no algorithm computing the weak or strong x-degeneracy of a graph G in $\mathcal{O}(n^{2-\varepsilon})$ time for arbitrary, but fixed $x \in \mathbb{N}_+$, unless the SETH breaks, even when G contains $\mathcal{O}(n \log n)$ edges, where $\varepsilon > 0$.*

As Theorem 5.11 holds even when the input graph contains $\mathcal{O}(n \log n)$ edges, we observe that the running time $\mathcal{O}(nm)$ of our algorithm computing the strong x-degeneracy of a graph as described in Lemma 5.7 is already optimal up to a poly-logarithmic factor.

Next, we will present our gap-framework for generalizing the KWB-algorithm (for solving Maximum Clique) to Maximum Clique relaxations.

5.2 The Gap-Framework

After we have formally generalized the standard degeneracy of a graph, we will next generalize the KWB-algorithm for Maximum Clique to provide a framework for finding a maximum-order relaxed clique in a graph. Recall from Section 5.1 that we study the weak x-degeneracy rather than the strong x-degeneracy of a graph, as the weak x-degeneracy is the smaller parameter.

We will show that the maximization problems of several clique relaxations Π can be solved in $\mathcal{O}^*(c^{\alpha_x+1-\omega_\Pi})$ time, where $x \in \mathbb{N}$ is an upper bound on the diameter of the solutions of the corresponding maximization problem, $c \in \mathbb{N}$ only depends on Π, and ω_Π is the maximum order of a subgraph which is a relaxed clique Π in the input graph G.

Recall that we say that $(\alpha_x + 1) - \omega_\Pi$ is a *gap*, as $\alpha_x + 1$ is an upper bound on the order of a subgraph satisfying Π.

Recall that the KWB-algorithm has two ingredients: First, we construct a Turing kernel for G with respect to the standard degeneracy of G, and afterwards we branch over two non-adjacent vertices. To generalize these ideas for clique relaxations, our framework replaces a standard degeneracy ordering of G with a weak x-degeneracy of G for some fixed $x \in \mathbb{N}_+$, and afterwards we branch over incompatible vertices (that is, vertices which cannot be part of the same solution). We will present our Turing kernelization in Section 5.2.1. Afterwards, we will recall the notion of incompatible vertices in Section 5.2.2 and generalize this idea to provide a simple branching algorithm for several clique relaxations. In Section 5.2.3, we will combine our Turing kernel and our branching algorithm to provide a generalization of the KWB-algorithm.

Next, we will formally define the problem we will solve.

Problem definitions. We say that a set Π of graphs is a *graph property.* If a graph G is contained in Π, then we say that G *satisfies* Π. As an example, "being a clique" is a graph property.

We briefly mention that there are clique relaxations which cannot be expressed with our notion of graph properties. A common example is the *s-clique* property, which can depend on vertices outside of the s-clique. These properties were called *weak* by Pattillo, Youssef, and Butenko [PYB13, Section 3.4], and we refer for more information on them to Komusiewicz [Kom16] and Pattillo, Youssef, and Butenko [PYB13]. We will need the "independence" on the graph outside of the solution to construct a Turing kernel in Lemma 5.14, which is a crucial part of our framework. Hence, we will not study weak properties in more detail.

Next, we define the decision problem as well as the maximization problem of finding a vertex set $S \subseteq V(G)$ such that $G[S] \in \Pi$. Note that we only consider induced subgraphs rather than all subgraphs, as we will only study clique relaxations in this thesis, thus adding more edges to a solution should never be an issue.

Induced Π-Subgraph [LY80]
Input: A graph property Π, a graph G and $k \in \mathbb{N}$.
Question: Is there a vertex set $S \subseteq V(G)$ such that $|S| \geq k$ and $G[S]$ satisfies Π?

Maximum Induced Π-Subgraph [LY80]
Input: A graph property Π and a graph G.
Question: A maximum-size vertex set $S \subseteq V(G)$ such that $G[S]$ satisfies Π.

If Π is fixed, then we say that the resulting problems are the *corresponding decision/maximization problems of* Π. Furthermore, for a given graph G, we say that $\omega_\Pi(G)$ is the maximum size of a set $S \subseteq V(G)$ such that $G[S] \in \Pi$. Note that (Maximum) Induced Π-Subgraph is a generalization of (Maximum) Clique.

In Sections 5.2.1 to 5.2.3, we will study the corresponding *decision* problems of particular graph properties in more detail. At the end of Section 5.2.3, we will show how to use all of our results to solve the corresponding *maximization* problems as well.

5.2.1 Turing Kernel

To generalize the first ingredient of the KWB-algorithm for Maximum Clique, we next show how to construct a Turing kernel for several Clique relaxations Π with respect to the weak x-degeneracy α_x of a graph, where each graph satisfying Π has diameter at most $x \in \mathbb{N}$.

Note that because every clique has diameter one, Clique admits a folklore Turing kernel where each input graph to the oracle contains at most $d + 1$ vertices, where d is the standard degeneracy of the graph. Next, we lift this idea to other graph properties Π as well. First, we assume that there is an upper bound $x \in \mathbb{N}$ on the diameter that a graph satisfying Π can have, where x only depends on Π. If there exists such an x for Π, then we say that Π *has diameter at most* x. Note that although many graph properties do not restrict the diameter of their graphs explicitly, graph properties like connected s-defective clique or connected s-plex still implicitly restrict the diameter of their graphs for all constant $s \in \mathbb{N}$, as we will show when applying our framework in Chapter 6. Additionally, we observe that if a disconnected graph satisfies Π, then we cannot find such an upper bound on the diameter. Thus, our framework will only apply to graph properties which exclude disconnected graphs.

First, recall that for the Turing kernel of Clique, each of the n input graphs to the oracle contains the closed right-neighborhood of some vertex with respect to a degeneracy ordering of the graph. Next, we will construct similar subgraphs for clique relaxations by using a weak x-degeneracy ordering and the closed weak x-right-neighborhoods of all vertices for some $x \in \mathbb{N}_+$. We start by introducing a new notion for these subgraphs.

Definition 5.12. Let G be a graph and $\preceq_{\alpha_x} = (v_1, v_2, \ldots, v_n)$ be a weak x-degeneracy ordering of G for some $x \in \mathbb{N}$. Then, $\mathcal{G}^x_{i,G,\preceq_{\alpha_x}}$ is the subgraph induced by the closed weak x-right-neighborhood $\mathrm{wrN}_x[v_i]$ of v_i, formally, $\mathcal{G}^x_{i,G,\preceq_{\alpha_x}} := G[\mathrm{wrN}_x[v_i]]$ for all $i \in [n]$.

If G, x, and $\preceq_{\alpha_x}$ are clear from the context, then we will omit their subscripts.

For a given weak x-degeneracy ordering $\preceq_{\alpha_x} = (v_1, v_2, \ldots, v_n)$ of G, we construct the vertex set and edge set of $\mathcal{G}^x_i$ by starting a depth-bounded breadth-first-search from the vertex v_i with $i \in [n]$, where we do not consider vertices left of v_i with respect to $\preceq_{\alpha_x}$. Hence, the construction of all $\mathcal{G}^x_i$ takes overall $\mathcal{O}(nm)$ time.

Proposition 5.13. *Let G be a graph and $\preceq_{\alpha_x} = (v_1, v_2, \ldots, v_n)$ be a weak x-degeneracy ordering of G for some $x \in \mathbb{N}$. Then one can construct the graphs $\mathcal{G}^x_1, \mathcal{G}^x_2, \ldots, \mathcal{G}^x_n$ according to Definition 5.12 in overall $\mathcal{O}(nm)$ time.*

We briefly mention that Walteros and Buchanan [WB20] presented a way to construct these subgraphs in overall $\mathcal{O}((n-d) \cdot d^2)$ time for the special case of $d = \alpha_1$. However, as we construct a weak x-degeneracy ordering of G in $\Omega(\alpha_x nm)$ time in order to construct these subgraphs for arbitrary $x \in \mathbb{N}$, we have not tried to optimize the running time of Proposition 5.13.

Next, we provide a Turing kernel for the corresponding decision problems of graph properties Π which have an upper bound on their diameter. This is a straightforward generalization of the folklore Turing kernel for Clique with respect to d and the Turing kernel of 2-Club with respect to Δ from Schäfer et al. [Sch+12]. For the sake of completeness, we provide a proof.

Lemma 5.14. *Let Π be a graph property such that Π has a diameter of at most x. Furthermore, let P be the corresponding decision problem of Π. Then, P admits a Turing kernel where each input graph to the oracle contains at most $\alpha_x + 1$ vertices, where α_x is the weak x-degeneracy of the input graph G. Furthermore, the Turing kernel is computable in $\mathcal{O}(\alpha_x nm)$ time.*

Proof. Let (G, k) be an instance of P. To construct the Turing kernel, first we compute a weak x-degeneracy ordering $\preceq_{\alpha_x} = (v_1, v_2, \dots, v_n)$ of G as described in Lemma 5.6. Next, we construct all subgraphs $\mathcal{G}_1^x, \mathcal{G}_2^x, \dots, \mathcal{G}_n^x$ with respect to $\preceq_{\alpha_x}$ as described in Proposition 5.13. Finally, we ask the oracle of the Turing kernel whether $(\mathcal{G}_i^x, k)$ is a YES-instance of P for all $i \in [n]$. We return YES if and only if the oracle returns YES at least once.

Next, we show the correctness of this approach, that is we show (G, k) is a YES-instance of P if and only if there exists some $i \in [n]$ such that $(\mathcal{G}_i^x, k)$ is a YES-instance of P.

"$\Rightarrow$": As (G, k) is a YES-instance of P, there exists a set $S \subseteq V(G)$ such that $|S| \geq k$ and $G[S]$ satisfies Π. Let v_i be the leftmost vertex of S with respect to $\preceq_{\alpha_x}$ for some $i \in [n]$. As $G[S]$ has diameter at most x, all vertices of S are part of the closed weak x-right-neighborhood of v_i. This means that $G[S] = \mathcal{G}_i^x[S]$ holds by definition of $\mathcal{G}_i^x$. Hence, as Π is a graph property, $(\mathcal{G}_i^x, k)$ is a YES-instance of P.

"$\Leftarrow$": As $(\mathcal{G}_i^x, k)$ is a YES-instance of P for some $i \in [n]$, there exists a set $S \subseteq V(\mathcal{G}_i^x)$ such that $|S| \geq k$ and $\mathcal{G}_i^x[S]$ satisfies Π. As $\mathcal{G}_i^x[S]$ is an induced subgraph of G, and Π is a graph property, (G, k) is a YES-instance of P.

Since one can, due to Lemma 5.6 and Proposition 5.13, compute a weak x-degeneracy ordering as well as our subgraphs in overall $\mathcal{O}(\alpha_x nm)$ time, and the oracle takes constant time to decide any of the n questions, the Turing kernel is computable in $\mathcal{O}(\alpha_x nm)$ time. Furthermore, by the definition of a weak x-degeneracy ordering, each input graph $\mathcal{G}_i^x$ to the oracle contains at most $\alpha_x + 1$ vertices. Hence, Lemma 5.14 holds. $\square$

To replace the oracle of the Turing kernel of Lemma 5.14 with an actual algorithm, we next introduce the notion of *incompatible vertices.*

5.2.2 Incompatible Vertices

To generalize the second ingredient of the KWB-algorithm for Maximum Clique, we next recall the notion of incompatible vertices from Komusiewicz et al. [Kom+19, Section 4.2]. First, we present a very broad definition of incompatible vertices, and show that restricting ourselves to so-called (c, p)-shrinkable graph properties leads to a simple branching algorithm.

There are many graph properties Π for which the corresponding decision problem can be solved by branching over two vertices which cannot be part of the same solution. For example, one can decide a Clique instance (G, k) as the survey Komusiewicz [Kom16, Chapter 1] presents it: If G is not a clique, then there must exist two non-adjacent vertices. As both of these vertices cannot be part of the same clique, we branch over removing one of these two vertices. As one can check whether G is a clique in polynomial time, or otherwise find two non-adjacent vertices in polynomial time as well, this leads to an algorithm deciding Clique in $\mathcal{O}^*(2^{n-k})$ time.

The idea of branching over two vertices which cannot be part of the same solution was also used before in the context of 2-club [BLP02; Kom+19; Sch+12]. To the best of our knowledge, Komusiewicz et al. [Kom+19, Section 4.2] were the first to formalize the notion of two vertices being *incompatible* in the context of 2-club variants. We will consider it here in a slightly more generalized form.

Definition 5.15 ([BLP02, Chapter 4], [Kom+19, Section 4.2])**.** Let Π be a graph property and G be a graph. A vertex set $I \subseteq V(G)$ is *incompatible* with respect to Π if no set $I \subseteq S \subseteq V(G)$ exists such that $G[S]$ satisfies Π.

Next, we show that incompatible vertices are useful for finding induced subgraphs of a graph G satisfying Π.

Proposition 5.16. *Let Π be a graph property and G be a graph. Then, G satisfies Π if and only if no set $I \subseteq V(G)$ of incompatible vertices in G exists.*

Proof. “$\Rightarrow$”: Assume towards a contradiction that there exists a set I of incompatible vertices. As $I \subseteq V(G)$, and G satisfies Π, this is a contradiction to the assumption that I is a set of incompatible vertices.

“$\Leftarrow$”: Assume towards a contradiction that G does not satisfy Π. As $V(G)$ already contains all vertices of G, it is by definition a set of incompatible vertices. This is a contradiction to the assumption that there is no set of incompatible vertices. □

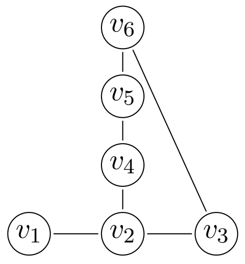
Proposition 5.16 already hints that computing a set I of incompatible vertices and branching over removing one of them might result in an algorithm for deciding the respective decision problems for several graph properties (like the algorithms for cliques and 2-clubs). However, we still have two issues when constructing these general branching algorithms. First, it might happen that it takes too much time to check whether G satisfies Π, which would slow down the whole branching algorithm. And second, it could happen that I has size n, which would lead to the intractable case of creating n branches. Hence, we will restrict the graph properties we study by defining a new feature for them.

Definition 5.17. Let Π be a graph property, $c \in \mathbb{N}$, and p be a polynomial. Furthermore, let G be an arbitrary graph. If it is decidable whether $G \in \Pi$ in $\mathcal{O}(p(n,m))$ time, and a set I of incompatible vertices with $|I| \leq c$ can be constructed in $\mathcal{O}(p(n,m))$ time in the case of $G \notin \Pi$, then Π is (c,p)*-shrinkable*.

Note that c and p (without its arguments) both depend only on Π, and *not* on the graph G. For example, we observe that Clique as well as 2-Club are both $(2, nm)$-shrinkable by our discussion at the beginning of Section 5.2.2.

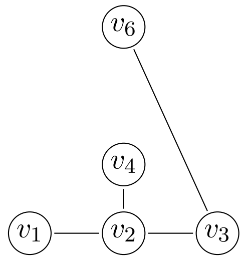
For the decision problems of (c,p)-shrinkable graph properties Π, it is easy to construct a branching algorithm for finding a set $S \subseteq V(G)$ of size at least $k \in \mathbb{N}$ in G such that $G[S] \in \Pi$, or `NONE` when G does not contain such a set:

- If $|V(G)| < k$, then we return `NONE`.
- If G satisfies Π, then we return $V(G)$.
- Otherwise, we compute a set of incompatible vertices I and branch over deleting one of the vertices in I. If one of the branches returns a set S, then we return S, otherwise we return `NONE`.



$C = \{\{v_1, v_5\}, \{v_1, v_6\}\}$

(a) A graph G as well as all pairs of incompatible vertices with respect to 2-Club.



$C' = \{\{v_1, v_6\}, \{v_4, v_6\}\}$

(b) $G' := G - v_5$ after removing v_5 from the set $\{v_1, v_5\}$ of incompatible vertices, as well as all pairs of incompatible vertices with respect to 2-Club.

Figure 5.2: Figure 5.2a is an illustration of a graph G with six vertices, and the set C of all pairs of incompatible vertices with respect to 2-Club. Recall that two vertices are incompatible with respect to 2-Club when they have distance at least three. Figure 5.2b is an illustration of G' after removing vertex v_5 from the incompatible set $\{v_1, v_5\}$ from G. We observe that in G', $\{v_4, v_6\}$ is a pair of incompatible vertices which was not part of C before. Hence, the task of removing vertices until the resulting graph does not contain a set of incompatible vertices anymore is *not* equivalent to the Hitting Set problem.

As the depth of our branching algorithm is bounded by $n - k$, the overall running time of our algorithm is $\mathcal{O}(c^{n-k} \cdot p(n, m))$. We will prove the correctness of this approach in Lemma 5.18. Note that the corresponding algorithms for clique and 2-club are the same ones we briefly discussed at the beginning of Section 5.2.2.

We briefly mention that one might think that our problem is a variant of the Hitting Set problem [GJ79], in the way that we are given an universe U with n elements and a collection $C \subseteq 2^U$, and we try to hit every $S \in C$ with at most $n - k$ elements. Here, U corresponds to $V(G)$ and a set $S \in C$ corresponds to a set of incompatible vertices in G. Although the comparison of both problems is appropriate, in our branching algorithm it can happen that by hitting one set, we create a new set of incompatible vertices which has not existed before, thus changing C. See Figure 5.2 for such an example for 2-Club. However, our problem is an *implicit* Hitting Set problem as stated by Moreno-Centeno and Karp [MK13, Chapter 1]. Here, C is not listed explicitly, but given implicitly as an oracle which also depends on the current solution set.

This difference of the two problems shows that we cannot lift the idea of the KWB-algorithm of Komusiewicz and Walteros and Buchanan where they searched for a large clique by searching for a small vertex cover in the complement graph, or in other words, solving a Hitting Set instance. We really have to branch over the current set of incompatible vertices and compute a new set of incompatible vertices afterwards as needed, instead of computing all sets of incompatible vertices at once.

Algorithm 1 Algorithm deciding INDUCED Π-SUBGRAPH

Input: A graph property Π which is (c, p)-shrinkable, an integer $x \in \mathbb{N}$ such that Π has diameter at most x, a graph G, and an integer $z \in \mathbb{N}$.

Output: A set $S \subseteq V(G)$ of size at least $\alpha_x(G) + 1 - z$ such that $G[S] \in \Pi$, or `NONE` when no such set exists.

```
1: function MAINΠ($\Pi, x, G, z$)
2:     $(v_1, v_2, \ldots, v_n) \leftarrow$ a weak $x$-degeneracy ordering of $G$     ▷ see Section 5.1.2
3:     Compute subgraphs $\mathcal{G}_1^x, \mathcal{G}_2^x, \ldots, \mathcal{G}_n^x$ of $G$     ▷ see Section 5.2.1
4:     Search for $S$ in our subgraphs with branching algorithm     ▷ see Section 5.2.2
5:     if suitable $S$ found then return $S$ else return NONE
6: end function
```

5.2.3 Combining Both Ingredients of the Framework

Next, we combine both ingredients of the KWB-algorithm for MAXIMUM CLIQUE, that is the Turing kernel as well as the branching algorithm, to decide INDUCED Π-SUBGRAPH for graph properties Π with the restrictions described in Sections 5.2.1 and 5.2.2. At the end, we will show how to use our results to solve the corresponding maximization problems as well.

In the following, we will slightly modify the notation for INDUCED Π-SUBGRAPH. First, we assume that the graph property Π has diameter at most x for some $x \in \mathbb{N}$. Then, instead of searching for a solution of order k in a graph G, we assume that an integer z is given such that we have to decide whether there is a vertex set $S \subseteq V(G)$ such that $|S| \geq \alpha_x + 1 - z$ holds and $G[S]$ satisfies Π, where α_x is the weak x-degeneracy of G. In other words, we check whether $g_\Pi(G) := \alpha_x + 1 - \omega_\Pi$ is at most z. If there exists such a set S, then we will return it, otherwise we will return `NONE`.

Note that if Π has diameter at most x, then $\alpha_x(G) + 1 - |S|$ is always non-negative. This is because from Lemma 5.14 we conclude that a set $S \subseteq V(G)$ such that $G[S] \in \Pi$ has order at most $\alpha_x(G) + 1$, as each subgraph in the Turing kernel contains at most $\alpha_x(G) + 1$ vertices.

By restricting ourselves to graph properties which are (c, p)-shrinkable for some $c \in \mathbb{N}$ and some polynomial p according to Definition 5.17, the following algorithm for INDUCED Π-SUBGRAPH combines our Turing kernel from Lemma 5.14 with our search-tree algorithm from Section 5.2.2 and the new parameter g_Π: Let (Π, x, G, z) be an instance of INDUCED Π-SUBGRAPH, where Π has diameter at most x and Π is (c, p)-shrinkable for some $c \in \mathbb{N}$ and a polynomial p. Recall that we check whether g_Π is at most z. First, we compute a weak x-degeneracy ordering of G as well as the subgraphs $\mathcal{G}_1^x, \mathcal{G}_2^x, \ldots, \mathcal{G}_n^x$ according to Definition 5.12. Afterwards, for each $\mathcal{G}_i^x$, we search a set S of size at least $\alpha_x(G) + 1 - z$ in $\mathcal{G}_i^x$ such that $\mathcal{G}_i^x[S] \in \Pi$ holds. This can be done by applying our branching algorithm from Section 5.2.2. If we find in any of these subgraphs such a set S, then g_Π is at most z and we return S. Otherwise, we return `NONE`. See Algorithm 1 for the pseudo-code.

Lemma 5.18. *Algorithm 1 is correct and can be executed in time $\mathcal{O}(c^z \cdot n \cdot p(n, m) + \alpha_x nm)$, where $z \in \mathbb{N}$ is a number for which we check whether $g_\Pi \leq z$, $c \in \mathbb{N}$, and p is a polynomial such that c and p only depend on Π.*

Proof. "$\Rightarrow$": Assume G contains a set $S \subseteq V(G)$ of size at least $\alpha_x(G) + 1 - z$ such that $G[S] \in \Pi$. Without loss of generality, assume that there is no set $S \subsetneq S' \subseteq V(G)$ such that $G[S']$ satisfies Π as well.

Let v_i be the leftmost vertex of S with respect to the computed weak x-degeneracy ordering. Due to Lemma 5.14, S is a valid solution for the subgraph $\mathcal{G}_i^x$.

Next, we show that for the subgraph $\mathcal{G}_i^x$, our branching algorithm will find S in one of its branches. If $S = V(\mathcal{G}_i^x)$, then our algorithm will return S. Otherwise, it will compute a set of incompatible vertices I and branch over removing one of these $|I|$ vertices. We observe that by the definition of an incompatible set, it cannot happen that I is a subset of S, because $\mathcal{G}_i^x[S]$ satisfies Π. Hence, there exists at least one vertex $v \in I \setminus S$ so that the new graph $\mathcal{G}_i^x - v$ still completely contains S. Now, by applying this argumentation recursively and knowing that S is not part of a strictly larger solution, there is at least one branch of our algorithm that removes all vertices from $V(\mathcal{G}_i^x) \setminus S$. As S has size at least $\alpha_x(G) + 1 - z$, our branching algorithm will not "cut" this particular branch because of its depth. Hence, our algorithm will return a set of size at least $\alpha_x(G)+1-z$.

"$\Leftarrow$": Assume that for some $i \in [n]$, our branching algorithm returned a set $S \subseteq V(\mathcal{G}_i^x)$ for the subgraph $\mathcal{G}_i^x$ of size at least $\alpha_x(G) + 1 - z$. As our algorithm does not remove solely edges in any step, $\mathcal{G}_i^x[S] \in \Pi$ holds. Due to Lemma 5.14, S is a valid solution in G of size $\alpha_x(G) + 1 - z$.

A weak x-degeneracy ordering of G as well as α_x are computable in $\mathcal{O}(\alpha_x nm)$ time as shown in Lemma 5.6. Furthermore, one can compute the subgraphs $\mathcal{G}_1^x, \dots, \mathcal{G}_n^x$ in $\mathcal{O}(nm)$ time as shown in Proposition 5.13. For some fixed $i \in [n]$, our branching algorithm runs in $\mathcal{O}(c^{n_{\mathcal{G}_i^x} - (\alpha_x(G)+1-z)} \cdot p(n,m)) \subseteq \mathcal{O}(c^{(\alpha_x(G)+1)-(\alpha_x(G)+1-z)} \cdot p(n,m)) = \mathcal{O}(c^z \cdot p(n,m))$ time. All in all, Algorithm 1 can be executed in overall $\mathcal{O}(\alpha_x nm + (n \cdot c^z \cdot p(n,m))) = \mathcal{O}(c^z \cdot n \cdot p(n,m) + \alpha_x nm)$ time. $\square$

By iteratively calling Algorithm 1 for $z = 0, 1, \dots$ until we find the first set S such that $G[S] \in \Pi$, we obtain our framework:

Theorem 5.19. *Let Π be a graph property with diameter at most $x \in \mathbb{N}$, and let Π be (c,p)-shrinkable for some $c \in \mathbb{N}$ and a polynomial p only depending on Π. Furthermore, let P be the corresponding maximization problem of Π. Then, P can be solved in $\mathcal{O}(c^{g_\Pi} \cdot n \cdot p(n,m) + \alpha_x nm)$ time, where $g_\Pi := \alpha_x + 1 - \omega_\Pi$, with α_x being the weak x-degeneracy of G, and ω_Π being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies Π.*

Note that we apply the "(c,p)-shrinkable-algorithm" only to the subgraphs of the Turing kernel, which have an upper bound of α_x+1 on the number of vertices. Hence, this sub-algorithm can also be executed in $\mathcal{O}(p(\alpha_x, \alpha_x^2))$ time rather than $\mathcal{O}(p(n,m))$ time. We will use this observation more in Chapter 6.

In Chapter 6, we will apply Theorem 5.19 to the graph properties s-club, connected s-defective clique, and connected s-plex for all constant $s \in \mathbb{N}$. Note that we will give an overview of values of the new "relaxed gaps" for these three graph properties in real-world graphs in Appendix A.

Chapter 6

Applying the Gap-Framework

In Chapter 5, we have presented a framework for solving the respective maximization problem of various clique relaxations. In this chapter, we will show how to apply our gap-framework to the maximization problems Maximum Connected s-Defective Clique, Maximum Connected s-Plex, and Maximum s-Club. We will denote their corresponding graph properties with $\Pi_{\text{C}s\text{-DC}}$, $\Pi_{\text{C}s\text{-P}}$, and $\Pi_{s\text{-Club}}$, respectively. In other words, a graph G satisfies $\Pi_{\text{C}s\text{-DC}}$ when G is connected and contains at most $s \in \mathbb{N}$ non-edges; $\Pi_{\text{C}s\text{-P}}$ when G is connected and has minimum degree at least $n - s$ with $s \in \mathbb{N}_+$; and $\Pi_{s\text{-Club}}$ when G has diameter at most $s \in \mathbb{N}_+$.

Recall from Theorem 5.19 that in order to apply our framework to some graph property Π, we have to show that

(1) Π has diameter at most $x \in \mathbb{N}$, that is, all graphs satisfying Π have diameter at most $x \in \mathbb{N}$ (for applying our Turing kernel from Section 5.2.1), and

(2) Π is (c, p)-shrinkable, that is, for some $c \in \mathbb{N}$ and polynomial p (without its arguments) only depending on Π, we can decide for some graph G in $\mathcal{O}(p(n, m))$ time whether G satisfies Π, or otherwise compute a set of at most c incompatible vertices (that is, they cannot be all part of the same solution) in $\mathcal{O}(p(n, m))$ time as well (for applying our branching algorithm from Section 5.2.2).

In Sections 6.1.1, 6.2.1 and 6.3.1, our goal is to show that $\Pi_{\text{C}s\text{-DC}}$, $\Pi_{\text{C}s\text{-P}}$, and $\Pi_{s\text{-Club}}$ all satisfy both ingredients of our framework.

For ingredient (1), we will present in Section 6.1.1 an upper bound $x \in \mathbb{N}$ on the diameter of $\Pi_{\text{C}s\text{-DC}}$ which is tight, that is, for all $s \in \mathbb{N}$ there exists a graph $G \in \Pi_{\text{C}s\text{-DC}}$ with diameter exactly x. We will provide similar tight results from the literature for $\Pi_{\text{C}s\text{-P}}$ and $\Pi_{s\text{-Club}}$ in the respective subsections.

For ingredient (2), we will show that:

- $\Pi_{\text{C}s\text{-DC}}$ is $(2 \cdot (s + 1), n^2)$-shrinkable for all $s \in \mathbb{N}$,
- $\Pi_{\text{C}s\text{-P}}$ is $(s + 1, n + m)$-shrinkable for all $s \in \mathbb{N}_+$, and
- $\Pi_{s\text{-Club}}$ is $(2, n^2)$-shrinkable for all $s \in \mathbb{N}_+$.

We will also show in Section 6.1.1 that under standard complexity assumptions, there is no $c \in \mathbb{N}$ and no polynomial p such that for all $s \in \mathbb{N}$, $\Pi_{\text{C}s\text{-DC}}$ is (c,p)-shrinkable. This means, informally speaking, that we cannot get rid of the dependence on the parameter s in the size of the incompatible set. In Section 6.2.1, we will provide a similar result for $\Pi_{\text{C}s\text{-P}}$ for all $s \in \mathbb{N}_+$. In contrast to these two results, we will show in Section 6.3.1 that such a c and p exist for $\Pi_{s\text{-Club}}$, as it is $(2,n^2)$-shrinkable for all $s \in \mathbb{N}_+$, hence we can get rid of the dependence on the parameter s for $\Pi_{s\text{-Club}}$.

Combining both ingredients, we will apply our gap-framework to these three clique relaxations. Before we present the results, recall from Definition 5.1 that for some $x \in \mathbb{N}_+$, a graph G has weak x-degeneracy $\alpha_x \in \mathbb{N}_+$ when α_x is the minimum number such that each subgraph of G contains a vertex with an x-neighborhood of size at most α_x.

Theorem 6.1. *Let $s \in \mathbb{N}$. Then,* Maximum Connected s-Defective Clique *is solvable in $\mathcal{O}((2\cdot(s+1))^{g_{Cs\text{-}DC}} \cdot n \cdot \alpha_*^2 + \alpha_* nm)$ time, where $g_{Cs\text{-}DC} := \alpha_* + 1 - \omega_{Cs\text{-}DC}$, with α_* being the weak $\left\lfloor \sqrt{2s+\frac{1}{4}} + \frac{1}{2} \right\rfloor$-degeneracy of G, and $\omega_{Cs\text{-}DC}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{Cs\text{-}DC}$.*

Theorem 6.2. *Let $s \in \mathbb{N}_+$. Then,* Maximum Connected s-Plex *is solvable in $\mathcal{O}((s+1)^{g_{Cs\text{-}P}} \cdot n \cdot \alpha_s^2 + \alpha_s nm)$ time, where $g_{Cs\text{-}P} := \alpha_s + 1 - \omega_{Cs\text{-}P}$, with α_s being the weak s-degeneracy of G, and $\omega_{Cs\text{-}P}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{Cs\text{-}P}$.*

Theorem 6.3. *Let $s \in \mathbb{N}_+$. Then,* Maximum s-Club *is solvable in $\mathcal{O}(2^{g_{s\text{-}Club}} \cdot n \cdot \alpha_s^3 + \alpha_s nm)$ time, where $g_{s\text{-}Club} := \alpha_s + 1 - \omega_{s\text{-}Club}$, with α_s being the weak s-degeneracy of G, and $\omega_{s\text{-}Club}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{s\text{-}Club}$.*

To complement these results, note that we present the values for the "relaxed gaps" in some real-world graphs in Appendix A.

Afterwards, in Sections 6.1.2 and 6.2.2, we will show for the graph properties $\Pi_{\text{C}s\text{-DC}}$ and $\Pi_{\text{C}s\text{-P}}$ that parts of our framework can easily be replaced by "similar" algorithms. Our goal is to provide adaptations of our framework which might perform better in practice when certain additional criteria are met. To be more precise, we will start by adapting ingredient (1) of our framework. That is, we will show that "large" connected s-defective cliques and "large" connected s-plexes have a "small" diameter. These observations will help us finding better upper bounds on the sizes of our Turing kernels for $\Pi_{\text{C}s\text{-DC}}$ and $\Pi_{\text{C}s\text{-P}}$. In contrast, we will prove in Section 6.3.2 that for "large" s-clubs, a better upper bound on the diameter does not exist. Moreover, for $\Pi_{\text{C}s\text{-DC}}$, we will show that we can also adapt ingredient (2) of our framework. That is, we will replace in Section 6.1.2 the branching algorithm of our framework with an alternative branching algorithm from the literature.

Next, we will apply our gap-framework to Maximum Connected s-Defective Clique.

6.1 Maximum Connected s-Defective Clique

In the following, we will apply Theorem 5.19 to the family of the Maximum Connected s-Defective Clique problems for all $s \in \mathbb{N}$. Recall that in a graph G, $S \subseteq V(G)$ is a connected s-defective clique if $G[S]$ is connected and contains at most s non-edges. We will show how to fulfill all requirements of Theorem 5.19 for the graph property of “being a connected s-defective clique” (in the following denoted with $\Pi_{\text{C}s\text{-DC}}$) for all s. Afterwards, we will show that “large” connected s-defective cliques have a “small” diameter and present two different ways on how to deal with “small” connected s-defective cliques. Furthermore, we will provide a second branching algorithm for solving CONNECTED s-DEFECTIVE CLIQUE and show how we can adapt our framework with it.

6.1.1 Application of the Framework

Recall that in order to apply our framework, we have to show that for all $s \in \mathbb{N}$ there exists an upper bound on the diameter of $\Pi_{\text{C}s\text{-DC}}$, and that each $\Pi_{\text{C}s\text{-DC}}$ is (c, p)-shrinkable for some $c \in \mathbb{N}$ and some polynomial p.

Upper bound on diameter. For each $s \in \mathbb{N}$, an upper bound of $s + 1$ on the diameter of $\Pi_{\text{C}s\text{-DC}}$ can be achieved due to Pattillo, Youssef, and Butenko [PYB13, Proposition 4] because each connected s-defective clique is a connected $s + 1$-plex as observed by Trukhanov et al. [Tru+13, Section 3.1] and Shirokikh [Shi13, Proposition 2.7].

We will show next that $\left\lfloor\sqrt{2s + \frac{1}{4}} + \frac{1}{2}\right\rfloor$ is also an upper bound on the diameter of a connected s-defective clique for all $s \in \mathbb{N}_+$. Note that $\left\lfloor\sqrt{2s + \frac{1}{4}} + \frac{1}{2}\right\rfloor$ is at most $s + 1$ for all $s \in \mathbb{N}$. Furthermore, we show that our upper bound is tight, which means that for each $s \in \mathbb{N}$, there exists a connected s-defective clique with diameter exactly $\left\lfloor\sqrt{2s + \frac{1}{4}} + \frac{1}{2}\right\rfloor$.

Lemma 6.4. *For all $s \in \mathbb{N}$, a graph G satisfying the graph property $\Pi_{Cs\text{-}DC}$ has diameter at most $\left\lfloor\sqrt{2s + \frac{1}{4}} + \frac{1}{2}\right\rfloor$. This upper bound is tight.*

Proof. Let G be a graph with diameter $x \in \mathbb{N}_+$ satisfying $\Pi_{\text{C}s\text{-DC}}$ for some $s \in \mathbb{N}$. Hence, there exist two vertices u, v in G such that u and v have distance exactly x. Thus, there exists an induced path P_{x+1} on $x + 1$ vertices in G.

Next, we calculate how many edges P_{x+1} misses to be a clique K_{x+1} on $x+1$ vertices. Note that this number is a lower bound on the number of missing edges in G. Recall that due to the definition of a connected s-defective clique, G misses at most s edges. As $|E(K_{x+1})| = \frac{x^2+x}{2}$ and $|E(P_{x+1})| = x$, the following inequality has to hold:

$$s \geq \frac{x^2}{2} - \frac{x}{2}. \tag{6.1}$$

Solving Inequality 6.1 like an equation for x leads to the two roots $x = \frac{1}{2} + \sqrt{\frac{1}{4} + 2s}$ and $x = \frac{1}{2} - \sqrt{\frac{1}{4} + 2s}$. We only concentrate on the first root, as for all $s \in \mathbb{N}$, the value of x for the second root is non-positive, hence the diameter x of the graph would be non-positive. Due to the first root, $x \leq \sqrt{2s + \frac{1}{4}} + \frac{1}{2}$ has to hold. As the diameter x of a graph is always an integer, we conclude that $x \leq \left\lfloor\sqrt{2s + \frac{1}{4}} + \frac{1}{2}\right\rfloor$.

Lastly, note that by construction, for a given $s \in \mathbb{N}$, a path on $\left\lfloor \sqrt{2s + \frac{1}{4}} + \frac{1}{2} \right\rfloor + 1$ vertices is a connected s-defective clique and has diameter exactly $\left\lfloor \sqrt{2s + \frac{1}{4}} + \frac{1}{2} \right\rfloor$. □

(c, p)-shrinkable. Next, we will show that for all $s \in \mathbb{N}$, $\Pi_{\text{C}s\text{-DC}}$ is $(2 \cdot (s + 1), n^2)$-shrinkable according to Definition 5.17. Our proof is based on ideas from Observation 4.2 and Proposition 4.3.

Lemma 6.5. *For all $s \in \mathbb{N}$, the graph property $\Pi_{Cs\text{-}DC}$ is $(2 \cdot (s + 1), n^2)$-shrinkable.*

Proof. Let G be a graph. First, we show that it is possible to decide whether G satisfies $\Pi_{\text{C}s\text{-DC}}$ in linear time. Recall that $G \in \Pi_{\text{C}s\text{-DC}}$ holds if and only if G is connected and contains at most s non-edges. Hence, it is enough to check that G is connected and count the number of edges in G, which can be done in overall linear time.

Second, we show that if $G \notin \Pi_{\text{C}s\text{-DC}}$, then it is possible to construct a set I of incompatible vertices so that $|I| \leq 2 \cdot (s + 1)$. If G is disconnected, then two vertices from different connected components form an incompatible set of size two. Otherwise, G misses at least $s+1$ edges. It is possible to find a set of $s+1$ missing edges in $\mathcal{O}(n^2)$ time by constructing an adjacency matrix first. As $s+1$ "edges" have at most $2\cdot(s+1)$ unique endpoints, and not all of these endpoints can be part of the same connected s-defective clique, this set of endpoints is a set of incompatible vertices.

All in all, $\Pi_{\text{C}s\text{-DC}}$ is $(2 \cdot (s + 1), n^2)$-shrinkable, and Lemma 6.5 holds. □

Next, we mention that under standard complexity assumptions, it is unlikely that there exist a constant $c \in \mathbb{N}$ and a polynomial p such that for all $s \in \mathbb{N}$, $\Pi_{\text{C}s\text{-DC}}$ is (c, p)-shrinkable. This means, informally speaking, that we cannot get rid of the dependence on the parameter s in the size of the incompatible set. To observe this, recall that s-Defective Clique is $W[1]$-hard with respect to $n - k$ [Cai08; GNW07], where k is the solution size. If $\Pi_{\text{C}s\text{-DC}}$ would be (c, p)-shrinkable for all $s \in \mathbb{N}$, then due to our results in Section 5.2.2, we would have shown that the decision problem Connected s-Defective Clique where s is part of the input is fixed-parameter tractable with respect to $n-k$. However, as one can easily reduce s-Defective Clique with respect to $n-k$ to Connected s-Defective Clique with respect to $n - k$ by adding a new vertex which is connected to all other vertices in the graph, under standard complexity assumptions this would lead to a contradiction to the result that s-Defective Clique is $W[1]$-hard with respect to $n - k$.

Applying our gap-framework. Next, we connect Lemmas 6.4 and 6.5 to apply Theorem 5.19 to $\Pi_{\text{C}s\text{-DC}}$ for all $s \in \mathbb{N}$. Note that due to our Turing kernel from Lemma 5.14, we apply Lemma 6.5 only to graphs which have at most $\alpha_{\left\lfloor \sqrt{2s+\frac{1}{4}}+\frac{1}{2} \right\rfloor} + 1$ vertices.

Theorem 6.1. *Let $s \in \mathbb{N}$. Then,* Maximum Connected s-Defective Clique *is solvable in $\mathcal{O}((2 \cdot (s + 1))^{g_{Cs\text{-}DC}} \cdot n \cdot \alpha_*^2 + \alpha_* nm)$ time, where $g_{Cs\text{-}DC} := \alpha_* + 1 - \omega_{Cs\text{-}DC}$, with α_* being the weak $\left\lfloor \sqrt{2s + \frac{1}{4}} + \frac{1}{2} \right\rfloor$-degeneracy of G, and $\omega_{Cs\text{-}DC}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{Cs\text{-}DC}$.*

In practice, one could try to improve the running time of this algorithm by first applying our two problem kernels from Chapter 4.

6.1.2 Adaptation of the Framework

In the following subsection, we will discuss how to modify our gap-framework with respect to Maximum Connected s-Defective Clique. That is, we first show how to reduce the size of our Turing kernel when we assume that a "large" connected s-defective clique exists inside the input graph. Afterwards, we will show how we can replace the branching algorithm of our gap-framework by applying ideas from the literature.

Modifying the Turing Kernel

As discussed, the upper bound on the diameter given in Lemma 6.4 is, for arbitrary n, tight. However, for large connected s-defective cliques, we can give a better upper bound. The following observation can be seen as a modification of the proof of Kosub [Kos04, Proposition 6.2.2], which is a proof for a similar statement about the diameter of an s-plex which was first observed by Seidman and Foster [SF78].

Observation 6.6. *Let $s \in \mathbb{N}$. Then, every graph $G \in \Pi_{Cs\text{-}DC}$ with at least $s+2$ vertices has diameter at most two.*

Proof. Assume towards a contradiction that G has diameter at least three. Hence, G contains two vertices u, v with distance exactly three. In other words, the edge $\{u, v\}$ is not part of G, and for each vertex $w \in V(G) \setminus \{u, v\}$, at least one of the edges $\{u, w\}, \{w, v\}$ cannot be part of G. This means that G misses at least $1 + |V(G) \setminus \{u, v\}| \geq 1 + ((s + 2) - 2) > s$ edges, which is a contradiction to the assumption that G satisfies $\Pi_{\mathrm{C}s\text{-}\mathrm{DC}}$. Hence, G has diameter at most two, and Observation 6.6 holds. $\square$

Note that Observation 6.6 strengthens a result of Pattillo, Youssef, and Butenko [PYB13, Proposition 8.d] (observed by Gschwind et al. [Gsc+20, Table 1]), as Pattillo, Youssef, and Butenko only showed that s-defective cliques of order at least $s + 2$ are *connected*. Furthermore, as $s + 2 \leq 2s + 1$ holds for all $s \in \mathbb{N}_+$, Observation 6.6 strengthens a result of Shirokikh [Shi13, Corollary 4.3], as he only showed that s-defective cliques of order at least $2s + 1$ have diameter at most two.

Next, we can adapt Theorem 6.1 for solving Maximum Connected s-Defective Clique when the input graph G contains a connected s-defective clique of order at least $s + 2$: As the diameter of a maximum-order connected s-defective clique in G is at most two due to Observation 6.6, we only have to consider the weak 2-degeneracy of G, rather than the weak $\left\lfloor\sqrt{2s + \frac{1}{4}} + \frac{1}{2}\right\rfloor$-degeneracy as in Theorem 6.1. In the following, we will call this new algorithm the *adapted framework*. However, in the general case one cannot assume that G contains such a large connected s-defective clique. Then, our adapted framework alone might not return a maximum-order connected s-defective clique. For example, if $s = 3$ and the input graph $G := P_4$ is a path on four vertices, then the maximum-order connected s-defective clique of G is G itself, but as G has diameter three, our adapted framework will not consider it as a solution. Hence, we will provide two strategies for solving Maximum Connected s-Defective Clique in the case of $\omega_{\mathrm{C}s\text{-}\mathrm{DC}}(G) \leq s + 1$.

Finding small solutions through brute-force. The following idea serves as a warm-up. The most obvious idea for finding a maximum-order connected s-defective clique of order at most $s+1$ in G is to use a brute-force algorithm which checks all induced subgraphs of order at most $s+1$ in $\mathcal{O}(n^{s+2} \cdot s^2)$ time.

Hence, by returning a maximum-order connected s-defective clique which was returned by our adapted framework or our brute-force algorithm, we get the following result.

Theorem 6.7. *Let $s \in \mathbb{N}$.* Maximum Connected s-Defective Clique *is solvable in $\mathcal{O}(((2 \cdot (s+1))^{\alpha_2+1-\omega_{Cs\text{-}DC}} \cdot n \cdot \alpha_2^2 + \alpha_2 nm) + n^{s+2} \cdot s^2)$ time, with α_2 being the weak 2-degeneracy of G, and $\omega_{Cs\text{-}DC}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{Cs\text{-}DC}$.*

Note that as Observation 6.6 does not assume that the s-defective clique is already connected and we consider *all* subsets of size at most $s+1$ in Theorem 6.7, we can use the same algorithm to solve the standard Maximum s-Defective Clique problem as well in the same running time.

One might try to find an algorithm for solving Maximum Connected s-Defective Clique in the case of $\omega_{Cs\text{-DC}} \leq s+1$ in $f(s,k,d) \cdot (n+m)^{\mathcal{O}(1)}$ time, where f is some computable function and d is the standard degeneracy of a graph. In other words, one might try to get rid of the dependence on the parameter s in the exponent of the $\mathcal{O}(n^{s+2} \cdot s^2)$-time algorithm, while using parameters which are already part of the exponential running time of our framework (k corresponds to $\omega_{Cs\text{-DC}}$, d is at most α_2). Unfortunately, due to Corollary 3.6, Connected s-Defective Clique is $W[1]$-hard with respect to $k+s+d$, even if $k \in \mathcal{O}(\sqrt{s})$. Hence, assuming that $\omega_{Cs\text{-DC}} \leq s+1$ does not help us to develop an FPT-algorithm for deciding Connected s-Defective Clique with respect to $k+s+d$ under standard complexity assumptions.

Changing the diameter before finding small solutions through brute-force. Our second idea for solving Maximum Connected s-Defective Clique when the input graph only contains "small" solutions builds upon our first idea. The difference is that before we find such a "small" solution by brute-force, first we re-define what a *small* solution is by changing the upper bound on the diameter for our adapted framework. Informally speaking, by not only searching for connected s-defective cliques of diameter at most two with our adapted framework, but searching for connected s-defective cliques of diameter at most $\nu > 2$, the maximum-order of a connected s-defective cliques which has diameter at least $\nu+1$ decreases from $s+1$ to some smaller number. Hence, the exponent of the brute-force algorithm decreases. Note that this idea circumvents the hardness result from above.

Theorem 6.8. *Let $s, \nu \in \mathbb{N}$. Then,* Maximum Connected s-Defective Clique *is solvable in $\mathcal{O}(((2 \cdot (s+1))^{\alpha_*+1-\omega_{Cs\text{-}DC}} \cdot n \cdot \alpha_*^2 + \alpha_* nm) + n^{\nu+1} \cdot \nu^2)$ time, with α_* being the weak $\left\lceil 1 + \frac{2 \cdot (s+1)}{\nu} \right\rceil$-degeneracy of G, and $\omega_{Cs\text{-}DC}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{Cs\text{-}DC}$.*

The main idea to prove Theorem 6.8 is that a "large" connected s-defective clique has to contain a "large" star. As stars have diameter at most two, the connected s-defective clique also has to have a "small" diameter. We start with an easy observation about the maximum degree Δ of a connected s-defective clique. Although it is straightforward, to the best of our knowledge it was not considered before in the literature.

Observation 6.9. *Let $s \in \mathbb{N}$. Then, for every graph $G \in \Pi_{Cs\text{-}DC}$ it holds that $\Delta > (n-1) - \frac{2\cdot(s+1)}{n}$.*

Proof. Assume towards a contradiction that $\Delta \leq (n-1) - \frac{2\cdot(s+1)}{n}$. Recall that by definition, a connected s-defective clique contains at least $\binom{n}{2} - s$ edges. Next, we count the number of edges in G.

$$\begin{aligned}
\binom{n}{2} - s &\leq |E(G)| \\
&\overset{\text{Handshaking Lemma}}{=} \frac{\sum_{v\in V(G)} \deg(v)}{2} \\
&\leq \frac{n\cdot\Delta}{2} \\
&\leq \frac{n\cdot((n-1) - \frac{2\cdot(s+1)}{n})}{2} \\
&= \binom{n}{2} - (s+1)
\end{aligned}$$

However, this is a contradiction, as $\binom{n}{2} - s > \binom{n}{2} - (s+1)$. Hence, our assumption was wrong and Observation 6.9 holds. $\square$

For finding an upper bound on the diameter of a connected s-defective clique, note that for every graph G and vertex v it holds that $G[N[v]]$ contains a star with v as the center and $N(v)$ as the leaves. Thus every graph contains a star of order $\Delta + 1$.

Lemma 6.10. *Let $s \in \mathbb{N}$. Then, for every graph $G \in \Pi_{Cs\text{-}DC}$ it holds that* $\text{diameter}(G) \leq \left\lceil 1 + \frac{2\cdot(s+1)}{n} \right\rceil$.

Proof. We conclude from Observation 6.9 that G contains a star K' of order strictly greater than $n - \frac{2\cdot(s+1)}{n}$. Hence, there are strictly less than $\frac{2\cdot(s+1)}{n}$ vertices in G which are not part of K'. As K' has diameter at most two, G is connected, and each vertex which is not part of K' can increase the diameter of G by at most one, it holds that $\text{diameter}(G) < 2 + \frac{2\cdot(s+1)}{n}$. As the diameter of a graph is always an integer, we observe that $\text{diameter}(G) \leq \left\lceil 2 + \frac{2\cdot(s+1)}{n} \right\rceil - 1 = \left\lceil 1 + \frac{2\cdot(s+1)}{n} \right\rceil$. $\square$

Finally, we observe that for fixed s, our upper bound on the diameter on a connected s-defective clique is non-increasing. Thus, our upper bound on the diameter of a connected s-defective clique of order $\nu \in \mathbb{N}$ is also an upper bound on the diameter of a connected s-defective clique of order $n \geq \nu$. By searching for large and small connected s-defective cliques separately as in Theorem 6.7, we have proven Theorem 6.8.

Theorem 6.8. *Let $s, \nu \in \mathbb{N}$. Then,* Maximum Connected s-Defective Clique *is solvable in $\mathcal{O}(((2 \cdot (s+1))^{\alpha_*+1-\omega_{Cs\text{-}DC}} \cdot n \cdot \alpha_*^2 + \alpha_* nm) + n^{\nu+1} \cdot \nu^2)$ time, with α_* being the weak $\left\lceil 1 + \frac{2\cdot(s+1)}{\nu} \right\rceil$-degeneracy of G, and $\omega_{Cs\text{-}DC}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{Cs\text{-}DC}$.*

We observe that Theorem 6.7 is *not* a special case of Theorem 6.8, as setting $s \geq 2, \nu := s+2$ for Lemma 6.10 results in an upper bound of three on the diameter of a connected s-defective clique of order at least ν, while due to Observation 6.6 such a connected s-defective clique has diameter at most two.

Note that Chen et al. [Che+21, Property 2] also found a relation of the order and the diameter of a connected s-defective clique. However, their result has an additional constraint on the minimum order of a connected s-defective clique. Furthermore, for a connected s-defective clique variant which is based on the density of the solution, Pattillo, Youssef, and Butenko [PYB13, Proposition 8] found an upper bound on the diameter of these solutions. Additionally, they showed that for each $n \in \mathbb{N}$ there exists a solution with a diameter matching their upper bound. However, we were unable to relate Lemma 6.10 to the results of Chen et al. and Pattillo, Youssef, and Butenko. Nevertheless, our result provides a different perspective for finding an upper bound on the diameter of a connected s-defective clique.

Modifying the Branching Algorithm

After we have modified the Turing kernel of our framework for Maximum Connected s-Defective Clique, we will replace next the branching algorithm of our gap-framework for this problem. Recall from Section 5.2.2 that when applying our framework, we usually branch over deleting one of the vertices of an incompatible set. However, in the special case of Maximum Connected s-Defective Clique, we can describe a different branching algorithm which also takes the edges of the graph into account. This algorithm was first described by Raman and Saurabh [RS08, Theorem 10] for the dual problem of s-Defective Clique, the Partial Vertex Cover problem. We will shortly recall the idea of this algorithm for Connected s-Defective Clique.

The basic idea of this algorithm is that for an instance (G, k, s) of Connected s-Defective Clique where $n \geq k$ and G is not already a connected s-defective clique, we branch for each missing edge $\{u, v\} \notin E(G)$ into three cases:

1. Assume that u is not part of the solution; proceed on $G - u$;
2. Assume that v is not part of the solution; proceed on $G - v$;
3. Assume that u and v *are* both part of the solution; decrease s by one; proceed on $G - \{u, v\}$.

Note that we can verify whether G is a connected s-defective clique or otherwise find a missing edge of G in overall $\mathcal{O}(n^2)$ time (see the details inside the proof of Lemma 6.5). As we decrease $n - k$ in the first two cases, and we decrease s in the third case, this branching algorithm runs in $\mathcal{O}(3^{(n-k)+s} \cdot n^2)$ time. The correctness follows directly from Raman and Saurabh [RS08, Theorem 10]. For a discussion on the relation between the running times of the "original" branching algorithm following from Lemma 6.5 and the "new" branching algorithm of Raman and Saurabh, we refer to Section 4.2.

By replacing our branching algorithm from Theorem 6.1 with this new branching algorithm described above, we get the following result.

Theorem 6.11. *Let $s \in \mathbb{N}$. Then,* Maximum Connected s-Defective Clique *is solvable in $\mathcal{O}(3^{g_{Cs\text{-}DC}+s} \cdot n \cdot \alpha_*^2 + \alpha_* nm)$ time, where $g_{Cs\text{-}DC} := \alpha_* + 1 - \omega_{Cs\text{-}DC}$, with α_* being the weak $\left\lfloor \sqrt{2s + \frac{1}{4}} + \frac{1}{2} \right\rfloor$-degeneracy of G, and $\omega_{Cs\text{-}DC}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{Cs\text{-}DC}$.*

Next, we will apply our gap-framework to Maximum Connected s-Plex.

6.2 Maximum Connected s-Plex

In the following, we will apply Theorem 5.19 to the family of the Maximum Connected s-Plex problems for all $s \in \mathbb{N}_+$. Recall that in a graph G, $S \subseteq V(G)$ is a connected s-plex when $G[S]$ is connected and $\delta(G[S]) \geq |S| - s$, where δ is the minimum degree. We will show how to fulfill all requirements of Theorem 5.19 for the graph property of "being a connected s-plex" (in the following denoted with $\Pi_{\text{C}s\text{-P}}$) for all s. Afterwards, we will note that "large" connected s-plexes have a "small" diameter and present three different ways on how to deal with "small" connected s-plexes.

6.2.1 Application of the Framework

Recall that in order to apply our framework, we have to show that for all $s \in \mathbb{N}_+$ there exists an upper bound on the diameter of $\Pi_{\text{C}s\text{-P}}$, and that each $\Pi_{\text{C}s\text{-P}}$ is (c, p)-shrinkable for some $c \in \mathbb{N}$ and some polynomial p.

Upper bound on diameter. We start with the following observation from the literature.

Observation 6.12 (Pattillo, Youssef, and Butenko [PYB13, Proposition 4])**.** *For all $s \in \mathbb{N}_+$, the graph property $\Pi_{Cs\text{-}P}$ has diameter at most s. This upper bound is tight.*

Note that tight means that for each $s \in \mathbb{N}_+$, there exists a graph G (a path on $s + 1$ vertices) satisfying $\Pi_{\text{C}s\text{-P}}$ and having diameter exactly s.

(c, p)-shrinkable. Next, we will show that for all $s \in \mathbb{N}_+$, $\Pi_{\text{C}s\text{-P}}$ is $(s + 1, n + m)$-shrinkable according to Definition 5.17. Our proof is based on ideas from Komusiewicz et al. [Kom+09, Theorem 6].

Lemma 6.13. *For all $s \in \mathbb{N}_+$, the graph property $\Pi_{Cs\text{-}P}$ is $(s + 1, n + m)$-shrinkable.*

Proof. Let G be a graph. First, we show that it is possible to decide whether G satisfies $\Pi_{\text{C}s\text{-P}}$ in linear time. Recall that $G \in \Pi_{\text{C}s\text{-P}}$ holds if and only if G is connected and $\delta(G) \geq n - s$. Hence, it is enough to check that G is connected and compute the minimum degree of G, which can be done in overall linear time.

Second, we show that if $G \notin \Pi_{\text{C}s\text{-P}}$, then it is possible to construct a set I of incompatible vertices so that $|I| \leq (s+1)$. If G is disconnected, then two vertices from different connected components form an incompatible set of size two. Otherwise, by the definition of $\Pi_{\text{C}s\text{-P}}$, G contains a vertex v such that $\deg(v) \leq n-(s+1)$. Let J be a set of exactly s vertices which are not adjacent to v, and $I := J \cup \{v\}$. As Komusiewicz et al. [Kom+09, Theorem 6] already observed, there cannot exist a set $I \subseteq S \subseteq V(G)$ such that $G[S] \in \Pi_{\text{C}s\text{-P}}$, as otherwise $\deg_{G[S]}(v) \leq |S|-(s+1)$ holds, contradicting the definition of a connected s-plex. It is possible to find a minimum-degree vertex v as well as s non-neighbors of v in linear time.

All in all, $\Pi_{\text{C}s\text{-P}}$ is $(s+1, n+m)$-shrinkable, and Lemma 6.13 holds. □

Next, we mention that under standard complexity assumptions, it is unlikely that there exist a constant $c \in \mathbb{N}$ and a polynomial p such that for all $s \in \mathbb{N}_+$, $\Pi_{\text{C}s\text{-P}}$ is (c,p)-shrinkable. Informally speaking, we cannot get rid of the dependence on the parameter s in the size of the incompatible set. To observe this, note that s-Plex is $W[2]$-complete with respect to $n-k$ [Fel+11], where k is the solution size. If $\Pi_{\text{C}s\text{-P}}$ would be (c,p)-shrinkable for all $s \in \mathbb{N}_+$, then due to our results in Section 5.2.2, we would have shown that the decision problem Connected s-Plex where s is part of the input is fixed-parameter tractable with respect to $n-k$. However, as one can easily reduce s-Plex with respect to $n-k$ to Connected s-Plex with respect to $n-k$ by adding a new vertex which is connected to all other vertices in the graph, under standard complexity assumptions this would lead to a contradiction to the result that s-Plex is $W[2]$-hard with respect to $n-k$.

Applying our gap-framework. Next, we connect Observation 6.12 and Lemma 6.13 to apply Theorem 5.19 to $\Pi_{\text{C}s\text{-P}}$ for all $s \in \mathbb{N}_+$. Note that due to our Turing kernel from Lemma 5.14, we apply Lemma 6.13 only to graphs which have at most $\alpha_s + 1$ vertices.

Theorem 6.2. *Let $s \in \mathbb{N}_+$. Then,* Maximum Connected s-Plex *is solvable in $\mathcal{O}((s+1)^{g_{Cs\text{-}P}} \cdot n \cdot \alpha_s^2 + \alpha_s nm)$ time, where $g_{Cs\text{-}P} := \alpha_s + 1 - \omega_{Cs\text{-}P}$, with α_s being the weak s-degeneracy of G, and $\omega_{Cs\text{-}P}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{Cs\text{-}P}$.*

One could try to improve the running time of this algorithm by first applying one of the two problem kernels of s-Plex with respect to the combined parameter $(n-k)+s$ from Fellows et al. [Fel+11, Theorem 1] and Moser, Niedermeier, and Sorge [MNS12, Theorem 1], which are computable in $\mathcal{O}(n^4 \cdot m)$ time and quadratic time, respectively.

6.2.2 Adaptation of the Framework

As discussed, the upper bound on the diameter given in Observation 6.12 is, for arbitrary n, tight. However, for large connected s-plexes, Seidman and Foster [SF78] gave a better upper bound.

Observation 6.14. *Let $s \in \mathbb{N}_+$. Then, every graph $G \in \Pi_{Cs\text{-}P}$ with at least $2s-1$ vertices has diameter at most two.*

Hence, we can adapt Theorem 6.2 for solving Maximum Connected s-Plex when the input graph G contains a connected s-plex of order at least $2s-1$. This idea was already briefly discussed by Trukhanov et al. [Tru+13, Subsection 3.3.3]: As the diameter of a maximum-order connected s-plex in G is at most two due to Observation 6.14, we only have to consider the weak 2-degeneracy of G, rather than the weak s-degeneracy as in Theorem 6.2. In the following, we will call this new algorithm the *adapted framework*. However, in the general case one cannot assume that G contains such a large connected s-plex. Then, our adapted framework alone might not return a maximum-order connected s-plex. For example, if $s = 3$ and the input graph $G := P_4$ is a path on four vertices, then the maximum-order connected s-plex of G is G itself, but as G has diameter three, our adapted framework will not consider it as a solution. Hence, we will provide three strategies for solving Maximum Connected s-Plex in the case of $\omega_{\text{C}s\text{-P}}(G) \leq 2s-2$.

Finding small solutions through brute-force. Like in the warm-up for connected s-defective cliques, the most obvious idea for finding a maximum-order connected s-plex of order at most $2s-2$ in G is to use a brute-force algorithm which checks all induced subgraphs of order at most $2s-2$ in $\mathcal{O}(n^{2s-1} \cdot s^2)$ time.

Hence, by returning a maximum-order connected s-plex which was returned by our adapted framework or our brute-force algorithm, we get the following result.

Theorem 6.15. *Let $s \in \mathbb{N}_+$. Then, the* Maximum Connected s-Plex *problem is solvable in $\mathcal{O}(((s+1)^{\alpha_2+1-\omega_{Cs\text{-}P}} \cdot n \cdot \alpha_2^2 + \alpha_2 nm) + n^{2s-1} \cdot s^2)$ time, with α_2 being the weak 2-degeneracy of G, and $\omega_{Cs\text{-}P}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{Cs\text{-}P}$.*

Note that as Observation 6.14 does not assume that the s-plex is already connected and we consider *all* subsets of size at most $2s-2$ in Theorem 6.15, we can use the same algorithm to solve the standard Maximum s-Plex problem as well in the same running time.

One might try to find an algorithm for solving Maximum Connected s-Plex in the case of $\omega_{\text{C}s\text{-P}} \leq 2s-2$ in $f(s,k,d) \cdot (n+m)^{\mathcal{O}(1)}$ time, where f is some computable function and d is the standard degeneracy of a graph. In other words, one might try to get rid of the dependence on the parameter s in the exponent of the $\mathcal{O}(n^{2s-1} \cdot s^2)$-time algorithm, while using parameters which are already part of the exponential running time of our framework (k corresponds to $\omega_{\text{C}s\text{-P}}$, d is at most α_2). Unfortunately, this is not possible under standard complexity assumptions, as we will present next. Recall that Koana, Komusiewicz, and Sommer [KKS20, Theorem 3.3] showed that s-Plex is $W[1]$-hard with respect to the combined parameter $k+s+d$. By studying their reduction in more detail, we observe that if the resulting s-Plex-instance (G,k,s) is a YES-instance, then G contains an s-plex of order k, but not of order $k+1$, hence $\omega_{\text{C}s\text{-P}} \leq k$. Furthermore, we can observe that $k \leq 2s-2$ holds for all $s \geq 4$. Lastly, we observe that each s-plex of order k in G has to be connected. Hence, Connected s-Plex is $W[1]$-hard with respect to the combined parameter $k+s+d$, even if $\omega_{\text{C}s\text{-P}} \leq 2s-2$. Thus, assuming that $\omega_{\text{C}s\text{-P}} \leq 2s-2$ does not help us to develop an FPT-algorithm for deciding Connected s-Plex with respect to $k+s+d$ under standard complexity assumptions.

Changing the diameter before finding small solutions through brute-force. Like for Connected s-Defective Clique, our second idea for solving Maximum Connected s-Plex when the input graph only contains "small" solutions builds upon our first idea. The difference is that before we find such a "small" solution by brute-force, we re-define what a *small* solution is by changing the upper bound on the diameter for our adapted framework. Informally speaking, by not only searching for connected s-plexes of diameter at most two with our adapted framework, but searching for connected s-plexes of diameter at most $\nu > 2$, the maximum-order of a connected s-plex which has diameter at least $\nu + 1$ decreases from $2s - 2$ to some smaller number. Hence, the exponent of the brute-force algorithm decreases. Note that this idea circumvents the hardness result from above. Before we present this idea, we briefly mention that the obvious idea would be to use our approach from Observation 6.9 again, thus finding an upper bound on the number of non-neighbors of a vertex in a connected s-plex G which depends both on s and n. However, by definition of an s-plex, this number only depends on s, thus we would get the same upper bound of $s + 1$ on the diameter of a connected s-plex as in Observation 6.12. Next, we present a different upper bound on the diameter of $\Pi_{\text{Cs-P}}$ which depends both on s and ν, based on a result from the literature.

Theorem 6.16. *Let $s \in \mathbb{N}_+, \nu \in \mathbb{N}$. Then,* Maximum Connected s-Plex *is solvable in $\mathcal{O}(((s+1)^{\alpha_*+1-\omega_{Cs\text{-}P}} \cdot n \cdot \alpha_*^2 + \alpha_* nm) + n^{\nu+1} \cdot \nu^2)$ time, with α_* being the weak $\max\{\left\lceil \frac{\nu}{\nu-s+1} \right\rceil, 3 \cdot (\left\lfloor \frac{\nu-z}{\nu-s+1} \right\rfloor - 1) + z, z \in \{0,1,2\}\}$-degeneracy of G, and $\omega_{Cs\text{-}P}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{Cs\text{-}P}$.*

Proof. First, note that Pattillo, Youssef, and Butenko [PYB13, Proposition 5.a] showed for all $s \in \mathbb{N}_+$ that each graph G satisfying $\Pi_{\text{Cs-P}}$ has diameter at most $f(s,n) := \max\{\left\lceil \frac{n}{n-s+1} \right\rceil, 3 \cdot (\left\lfloor \frac{n-z}{n-s+1} \right\rfloor - 1) + z, z \in \{0,1,2\}\}$.

Note that we cannot guess the exact maximum-order of a connected s-plex in the input graph G in polynomial time (as Connected s-Plex is NP-hard), but we can only find a (heuristic) lower bound $\nu \in \mathbb{N}$ on this order. Hence, to use the result of Pattillo, Youssef, and Butenko for Theorem 6.16, we have to show that for all constant s it holds that f is non-increasing in order to know that $f(s,\nu)$ is an upper bound on the diameter we have to consider, even if the maximum order of a connected s-plex is larger than ν. Without loss of generality, we assume that $n \geq s+2$, as each connected subgraph of order at most $s+1$ is a connected s-plex [PYB13].

To show that $f(s,n) \geq f(s,n+1)$ holds, note that due to the definition of the max-function, it is sufficient to show that every function which is part of the max-function is itself non-increasing. Additionally, due to applying equivalent transformations, it is sufficient to show that $\frac{n-z}{n-s+1}$ is non-increasing for all $z \in \{0,1,2\}$. Next, we observe that

$$\frac{n-z}{n-s+1} = \frac{n-(s-1)+(s-1)-z}{n-s+1} = 1 + \frac{s-z-1}{n-s+1}.$$

It is easy to see that the denominator of $\frac{s-z-1}{n-s+1}$ is positive as we assume $n \geq s$. Furthermore, if $s \geq 3$, then the numerator is non-negative for all z, hence in this case it is easy to see that $1 + \frac{s-z-1}{n-s+1}$ is non-increasing.

For the case of $s = 2$, note that if $z \in \{0, 1\}$, then $1 + \frac{s-z-1}{n-s+1}$ is also non-increasing. In the case of $z = 2$, it is easy to see that $1 - \frac{1}{n-1} \in [0, 1)$ for all $n \geq s$. As the values get floored afterwards, $\left\lfloor 1 - \frac{1}{n-1} \right\rfloor$ is non-increasing for $n \geq s$.

For the case of $s = 1$, note that if $z = 0$, then $1 + \frac{s-z-1}{n-s+1}$ is also non-increasing. In the case of $z = 1$, it is easy to see that $1 - \frac{1}{n} \in [0, 1)$ for all $n \geq s$. As the values get floored afterwards, $\lfloor 1 - \frac{1}{n} \rfloor$ is non-increasing for $n \geq s$. In the case of $z = 2$, it is easy to see that $1 - \frac{2}{n} \in [0, 1)$ for all $n \geq s + 1$. As the values get floored afterwards, $\lfloor 1 - \frac{2}{n} \rfloor$ is non-increasing for $n \geq s$.

Hence, $f(s, n) \geq f(s, n + 1)$ holds for all $s \in \mathbb{N}_+, n \geq s + 2$.

In the end, we adapt our framework in the same way as in Theorem 6.8 for connected s-defective cliques. That is, we search for large and small connected s-plexes separately as in Theorem 6.15. Additionally, we assume that if the branching algorithm of our framework as well as the brute-force algorithm return a connected s-plex of size at most s, then it is possible to find a connected s-plex of order $s + 1$ in linear time. All in all, Theorem 6.16 holds. □

Note that if we set $\nu := 2s - 1$, then we can observe that Theorem 6.15 is a special case of Theorem 6.16. This is no surprise, as Pattillo, Youssef, and Butenko [PYB13, Proposition 5.a] showed that their formula is tight, that is for all $s \in \mathbb{N}_+, n \in \mathbb{N}$ there exists a graph $G \in \Pi_{\text{C}s\text{-P}}$ such that diameter$(G) = f(s, n)$.

We mention that Xiao et al. [Xia+17, Property 3] also found a relation of the order and the diameter of a connected s-plex by generalizing Observation 6.12. However, as in their formula the order and the diameter are more intertwined, we were unable to apply or relate their result to our framework.

Finding small solutions by iteratively increasing the search space. Our third idea for solving Maximum Connected s-Plex in the case of $\omega_{\text{C}s\text{-P}} \leq 2s - 2$ is to successively assume that $\omega_{\text{C}s\text{-P}} = 2s - 2, 2s - 3, \ldots, 0$. Let z be the value of the current iteration. In each iteration, we assume that the diameter of a connected s-plex is at most $z - 1$, hence we could adapt Theorem 6.2 as before, but this time we consider the weak $(z - 1)$-degeneracy rather than the weak s-degeneracy. We would return the first connected s-plex S we found such that $|S| \geq z$. However, as the upper bound of $z - 1$ on the diameter decreases in each iteration, it would be sufficient to return the maximum-order connected s-plex found in the case of $z = 2s - 2$. Furthermore, as $(2s - 2) - 1 \geq s$ for all $s \geq 3$, this idea alone would result in an algorithm which would not perform better than the original algorithm of Theorem 6.2.

Hence, to make our idea useful, we have to find another upper bound on the diameter of a connected s-plex S that is non-decreasing when the order of S decreases. Furthermore, this upper bound on the diameter should also be smaller than the trivial upper bound of s from Observation 6.12. Next, we present a result from the literature. Due to Kosub [Kos04, Propostion 6.2.2], for all $s \in \mathbb{N}_+$ it holds that if a connected s-plex S is of order at most $2s - 2$, then it has diameter at most $2s + 2 - |S|$. Note that if $\omega_{\text{C}s\text{-P}} \geq s + 2$ holds for the input graph G, then the upper bound of Kosub on the diameter of a maximum-order connected s-plex in G is at most the trivial upper bound of $\omega_{\text{C}s\text{-P}} - 1$. Furthermore, note that Maximum Connected s-Plex is trivially solvable for $\omega_{\text{C}s\text{-P}} \leq s+1$ by returning any connected subgraph of order exactly $s+1$ as observed by Pattillo, Youssef, and Butenko [PYB13, Section 5].

Hence, the following algorithm can be applied to solve MAXIMUM CONNECTED s-PLEX for some $s \in \mathbb{N}_+$ and input graph G:

- Execute our adapted framework, if it returns a connected s-plex S of order at least $2s-1$, then return S.
- Search for a connected s-plex S of order at least z with respect to the weak $(2s+2-z)$-degeneracy of G for all $z = 2s-2, 2s-3, \dots, \omega_{\text{C}s\text{-P}}$; return the first S such that $|S| = z$.

As the formula of Kosub is non-decreasing for constant s and decreasing $|S|$, and $2s+2-z \geq 2$ holds for all $s \in \mathbb{N}_+, z \in [2s-2]$, it holds that $2s+2-\omega_*$ is an upper bound on the largest diameter we have to consider, where $\omega_* := \min\{2s-1, \omega_{\text{C}s\text{-P}}\}$. Hence, we get the following result.

Theorem 6.17. *Let $s \in \mathbb{N}_+$. Then, the* MAXIMUM CONNECTED s-PLEX *problem is solvable in $\mathcal{O}((2s-\omega_*)\cdot((s+1)^{\alpha_*+1-\omega_{Cs\text{-}P}}\cdot n\cdot\alpha_*^2+\alpha_* nm))$ time, with α_* being the weak $(2s+2-\omega_*)$-degeneracy of G, $\omega_{Cs\text{-}P}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{Cs\text{-}P}$, and $\omega_* := \min\{2s-1, \omega_{Cs\text{-}P}\}$.*

In practice, this algorithm might perform better than the two brute-force algorithms before in the case of $\omega_{\text{C}s\text{-P}}$ being slightly smaller than $2s-1$. In other words, in this case the diameter of a maximum-order connected s-plex is not necessarily two, but still does not depend on s.

Finally, we will apply our gap-framework to MAXIMUM s-CLUB.

6.3 Maximum s-Club

In the following, we will apply Theorem 5.19 to the family of the MAXIMUM s-CLUB problems for all $s \in \mathbb{N}_+$. Recall that in a graph G, $S \subseteq V(G)$ is an s-club when $G[S]$ has diameter at most s. We will show how to fulfill all requirements of Theorem 5.19 for the graph property of "being an s-club" (in the following denoted with $\Pi_{s\text{-Club}}$) for all s. Afterwards, in contrast to connected s-defective cliques and connected s-plexes, we will show for the sake of completeness that assuming that an s-club is "large" does not result in a "small" diameter of this s-club.

6.3.1 Application of the Framework

To give another example on how to apply our framework from Theorem 5.19, we will provide all of the steps for applying it to $\Pi_{s\text{-Club}}$. Recall that in order to apply our framework, we have to show that for all $s \in \mathbb{N}_+$ there exists an upper bound on the diameter of $\Pi_{s\text{-Club}}$, and that each $\Pi_{s\text{-Club}}$ is (c,p)-shrinkable for some $c \in \mathbb{N}$ and some polynomial p.

The upper bound of s on the diameter follows directly from the definition.

Observation 6.18. *For all $s \in \mathbb{N}_+$, the graph property $\Pi_{s\text{-Club}}$ has diameter at most s. This upper bound is tight.*

Note that tight means that for each $s \in \mathbb{N}_+$, there exists a graph G (a path on $s+1$ vertices) satisfying $\Pi_{s\text{-Club}}$ and having diameter exactly s.

(c,p)-shrinkable. Next, we will show that for all $s \in \mathbb{N}_+$, $\Pi_{s\text{-Club}}$ is $(2, nm)$-shrinkable according to Definition 5.17. Our proof is based on ideas from Schäfer et al. [Sch+12, Theorem 5].

Lemma 6.19. *For all $s \in \mathbb{N}_+$, the graph property $\Pi_{s\text{-Club}}$ is $(2, nm)$-shrinkable.*

Proof. Let G be a graph. First, we have to show that it is possible to decide whether G satisfies $\Pi_{s\text{-Club}}$ in $\mathcal{O}(nm)$ time. Second, we have to show that if $G \notin \Pi_{s\text{-Club}}$, then it is possible to construct a set I of incompatible vertices in $\mathcal{O}(nm)$ time so that $|I| = 2$.

Both can be done by computing the distances between all pairs of vertices by starting a breadth-first-search from each vertex in overall $\mathcal{O}(nm)$ time. Recall that $G \in \Pi_{s\text{-Club}}$ holds if and only if G has diameter at most s. Hence by the definition of the diameter of a graph, $G \in \Pi_{s\text{-Club}}$ holds if and only if all pairs of vertices have distance at most s. Second, if $G \notin \Pi_{s\text{-Club}}$, then it is easy to see that there have to exist two vertices u, v in G with distance at least $s+1$. As both u and v cannot be part of the same s-club, they form a set I of incompatible vertices of size two. All of this can be done in overall $\mathcal{O}(nm)$ time.

All in all, $\Pi_{s\text{-Club}}$ is $(2, nm)$-shrinkable, and Lemma 6.19 holds. □

We mention that showing $\Pi_{s\text{-Club}}$ is $(2, n^2)$-shrinkable for all $s \in \mathbb{N}_+$ would mean that one could compute the diameter of a graph G in $\mathcal{O}(\text{diameter}(G) \cdot n^2)$ time by iteratively calling this "$(2, n^2)$-shrinkable-algorithm" for all $s = 1, 2, \ldots, \text{diameter}(G)$, until no set of incompatible vertices exists anymore. Such an $\mathcal{O}(\text{diameter}(G) \cdot n^2)$-time algorithm would improve the best known algorithm for computing the diameter of a graph. Although not relevant for following the content of this thesis, we refer to Bentert and Nichterlein [BN19] for a discussion on algorithms for computing the diameter of a graph.

Applying our gap-framework. Next, we connect Observation 6.18 and Lemma 6.19 to apply Theorem 5.19 to $\Pi_{s\text{-Club}}$ for all $s \in \mathbb{N}_+$. Note that due to our Turing kernel from Lemma 5.14, we apply Lemma 6.19 only to graphs which have at most $\alpha_s + 1$ vertices.

Theorem 6.3. *Let $s \in \mathbb{N}_+$. Then,* MAXIMUM s-CLUB *is solvable in $\mathcal{O}(2^{g_{s\text{-Club}}} \cdot n \cdot \alpha_s^3 + \alpha_s nm)$ time, where $g_{s\text{-Club}} := \alpha_s + 1 - \omega_{s\text{-Club}}$, with α_s being the weak s-degeneracy of G, and $\omega_{s\text{-Club}}$ being the maximum size of a set $S \subseteq V(G)$ such that $G[S]$ satisfies $\Pi_{s\text{-Club}}$.*

Note that Hartung, Komusiewicz, and Nichterlein [HKN15a, Corollary 3] proved that under the assumption that the SETH (see Hypothesis 5.8) holds, s-CLUB cannot be decided in $\mathcal{O}^*((2-\varepsilon)^{n-\omega_{s\text{-Club}}})$ time for all $s \geq 2$. As all subgraphs we look at have at most $\alpha_s + 1$ vertices, the result of Hartung, Komusiewicz, and Nichterlein implies that Theorem 6.3 cannot be significantly improved without violating the SETH.

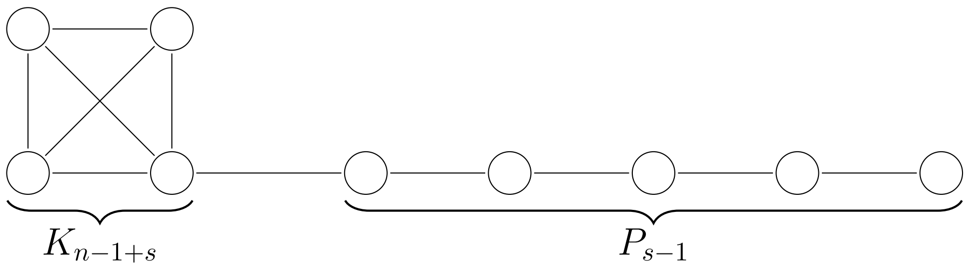


Figure 6.1: An illustration of our graph $G_{s,n}$ with $s \in \mathbb{N}_+, n \geq s+1$. Note that $G_{s,n}$ has diameter exactly s.

6.3.2 No Adaptation of Framework Possible

As discussed, the upper bound on the diameter given in Observation 6.18 is, for arbitrary n, tight. Unfortunately, in contrast to connected s-defective cliques and connected s-plexes, there is no better upper bound for large s-clubs, as we will show next.

We observe that for all $s, n \in \mathbb{N}_+$ such that $n \geq s+1$, there exists an s-club of order n and diameter exactly s.

Observation 6.20. *Let $s \in \mathbb{N}_+$, $n \geq s+1$. Then, there exists a graph $G_{s,n} \in \Pi_{s\text{-Club}}$ of order n such that G has diameter exactly s.*

Proof. Let $G_{s,n}$ be a graph consisting of the disjoint union of a clique K_{n+1-s} and a path P_{s-1}. Furthermore, if $s \geq 2$, then we add an edge connecting an arbitrary vertex of K_{n+1-s} with one of the endpoints of P_{s-1}. See Figure 6.1 for an illustration of $G_{s,n}$. It is easy to see that $G_{s,n}$ has order exactly n and diameter exactly s. □

In the context of this thesis, in this chapter we have presented how to apply our gap-framework from Theorem 5.19 (which is a generalization of the KWB-algorithm for solving Maximum Clique) to some Maximum Clique relaxations. Hence, the "relaxed gaps" $g_{\text{C}s\text{-DC}}$, $g_{\text{C}s\text{-P}}$, and $g_{s\text{-Club}}$ can be seen as a middle ground between the "small" clique-core gap $d+1-\omega$ and the "large" gap $n-\omega$.

In Appendix A, we provide the values of these relaxed gaps in many real-world graphs. Note that Walteros and Buchanan [WB20, Table 1] observed that if a Maximum Clique real-world instance is hard to solve *for all solvers*, then its clique-core gap is large. With the data provided in Appendix A, one could investigate whether a similar observation holds for some Maximum Clique relaxations and their respective "relaxed gaps".

Chapter 7

Conclusion

We studied the KWB-algorithm of Komusiewicz [Kom11, Proposition 5.4] and Walteros and Buchanan [WB20, Theorem 1] for solving Maximum Clique and generalized it to Maximum Clique relaxations. It turned out that the KWB-algorithm is not trivially generalizable to Maximum Clique relaxations, as the decision problems s-Defective Clique, s-Plex [KKS20], and s-Club [Har+15] are all $W[1]$-hard with respect to the combined parameter clique-core gap plus "relaxation parameter", $g + s$, where $g := d + 1 - \omega$ is the gap between the maximum possible order "degeneracy plus one" of a clique in G and the actual order ω of a clique in a graph. To circumvent this hardness result, we studied the larger gap between the order n of the input graph G and the maximum-order ω_Π of a relaxed clique in G, $n - \omega_\Pi$, and saw that s-Defective Clique [RS08], s-Plex [Kom+09], and s-Club [Sch+12] are all fixed-parameter tractable with respect to the combined parameter $(n - \omega_\Pi) + s$. Furthermore, we provided a new branching algorithm and two problem kernels for s-Defective Clique with respect to $(n - k) + s$, where k is the solution size. Afterwards, we studied the ingredients of the KWB-algorithm in more detail to construct our *gap-framework* which can be used to solve Maximum Clique relaxations. To find a "middle ground" between the "small" clique-core gap and the "large" gap $n - \omega$, we introduced and studied for $x \in \mathbb{N}_+$ the *weak and strong x-degeneracies* α_x, β_x of a graph, which both generalize the standard degeneracy. Next, we applied our gap-framework to Maximum (Connected) s-Defective Clique, Maximum (Connected) s-Plex, and Maximum s-Club. We also provided the values of the weak and strong x-degeneracies and the "relaxed gaps" $g_\Pi := \alpha_x + 1 - \omega_\Pi$ for Maximum Connected s-Defective Clique, Maximum Connected s-Plex, and Maximum s-Club in many real-world graphs.

Future research directions. First, we repeat the open questions we have already summarized in Table 1.2 on page 19. It is open whether (Connected) s-Plex and (Connected) s-Defective Clique are fixed-parameter tractable with respect to the combined parameter $g_2 := \alpha_2 + 1 - \omega_\Pi$ plus the respective relaxation parameter s. Furthermore, it is open whether s-Plex and s-Defective Clique are fixed-parameter tractable with respect to the combined parameter $g_s := \alpha_s + 1 - \omega_\Pi$ plus s.

Next, it would be interesting to implement the algorithms we provided for Maximum (Connected) s-Defective Clique, Maximum (Connected) s-Plex, and Maxi-

MUM s-CLUB in Chapter 5, and show how they behave for different classes of graphs, such as random graphs or real-world graphs. Note that the KWB-algorithm, which we generalized, is a state-of-the-art algorithm for solving MAXIMUM CLIQUE on real-world graphs [WB20]. Furthermore, for solving MAXIMUM 2-CLUB, Hartung, Komusiewicz, and Nichterlein [HKN15a] already implemented and studied an algorithm which also generates a Turing kernel where each subgraph is solved with a branching algorithm, which is similar to our gap-framework. However, in contrast to our gap-framework, their orderings on solving the subgraphs of the Turing kernel are heuristic and do not provide any worst-case running-time improvements. It would be interesting to compare their orderings to our weak and strong x-degeneracy orderings. Additionally, it would be interesting to implement both of our problem kernels for s-DEFECTIVE CLIQUE with respect to $(n-k)+s$ and study their usefulness in practice.

Applying the gap-framework to *all* clique relaxations as presented in Table 1.1 on page 10 was out of scope of this thesis. However, we guess that the gap-framework is applicable to many of these clique relaxations as well, as many of these implicitly have an upper bound on the diameter of their solutions. For example, Pattillo, Youssef, and Butenko [PYB13] also provided an upper bound on the diameter of *s-blocks*.

Lastly, we observe from Tables A.1 to A.4 that for all $x \in \mathbb{N}_x$ it takes much time to compute a weak x-degeneracy ordering of a graph with our algorithm as described in Lemma 5.6. Recall from Theorem 5.11 that a weak x-degeneracy ordering for arbitrary $x \geq 2$ and all $\varepsilon > 0$ is not computable in $\mathcal{O}(n^{2-\varepsilon})$ time, unless the SETH [IPZ01] breaks, which would result in a major breakthrough in parameterized complexity theory. One way to circumvent this hardness result would be to *approximate* the weak x-degeneracy of a graph. For the standard degeneracy of a graph, this topic has already been researched in the context of streaming large graphs [FT14; FT16]. It would be interesting to study whether similar techniques can be applied to the weak x-degeneracy.

Literature

[AC12] M. T. Almeida and F. D. Carvalho. "Integer models and upper bounds for the 3-club problem". In: *Networks* 60.3 (2012), pp. 155–166. URL: https://doi.org/10.1002/net.21455 (cit. on p. 16).

[AFS11] O. Amini, F. V. Fomin, and S. Saurabh. "Implicit branching and parameterized partial cover problems". In: *Journal of Computer and System Sciences* 77.6 (2011), pp. 1159–1171. URL: https://doi.org/10.1016/j.jcss.2010.12.002 (cit. on p. 13).

[ARS02] J. Abello, M. G. C. Resende, and S. Sudarsky. "Massive quasi-clique detection". In: *LATIN 2002: Theoretical Informatics, 5th Latin American Symposium, Cancun, Mexico, 2002*. Vol. 2286. Lecture Notes in Computer Science. Springer, 2002, pp. 598–612. URL: https://doi.org/10.1007/3-540-45995-2_51 (cit. on p. 10).

[BB98] N. H. Bshouty and L. Burroughs. "Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem". In: *STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science, 1998, Proceedings*. Vol. 1373. Lecture Notes in Computer Science. Springer, 1998, pp. 298–308. URL: https://doi.org/10.1007/BFb0028569 (cit. on pp. 13, 23, 28).

[BBH11] B. Balasundaram, S. Butenko, and I. V. Hicks. "Clique relaxations in social network analysis: the maximum k-plex problem". In: *Operations Research* 59.1 (2011), pp. 133–142. URL: https://doi.org/10.1287/opre.1100.0851 (cit. on pp. 14, 15).

[BBP06] V. Boginski, S. Butenko, and P. M. Pardalos. "Mining market data: A network approach". In: *Computers & Operations Research* 33.11 (2006), pp. 3171–3184. URL: https://doi.org/10.1016/j.cor.2005.01.027 (cit. on p. 9).

[BBT05] B. Balasundaram, S. Butenko, and S. Trukhanov. "Novel approaches for analyzing biological networks". In: *Journal of Combinatorial Optimization* 10.1 (2005), pp. 23–39. URL: https://doi.org/10.1007/s10878-005-1857-x (cit. on p. 16).

[Ben+19] M. Bentert, A. Himmel, H. Molter, M. Morik, R. Niedermeier, and R. Saitenmacher. "Listing all maximal k-plexes in temporal graphs". In: *ACM Journal of Experimental Algorithmics* 24.1 (2019), 1.13:1–1.13:27. URL: https://doi.org/10.1145/3325859 (cit. on pp. 15, 16).

[BG93] J. F. Buss and J. Goldsmith. “Nondeterminism within P”. In: *SIAM Journal on Computing* 22.3 (1993), pp. 560–572. URL: https://doi.org/10.1137/0222038 (cit. on p. 41).

[BGP13] H. Broersma, P. A. Golovach, and V. Patel. “Tight complexity bounds for FPT subgraph problems parameterized by the clique-width”. In: *Theoretical Computer Science* 485 (2013), pp. 69–84. URL: https://doi.org/10.1016/j.tcs.2013.03.008 (cit. on p. 13).

[BH03] G. D. Bader and C. W. V. Hogue. “An automated method for finding molecular complexes in large protein interaction networks”. In: *BMC Bioinformatics* 4 (2003), p. 2. URL: https://doi.org/10.1186/1471-2105-4-2 (cit. on pp. 10, 11, 16).

[Bin+12] D. Binkele-Raible, H. Fernau, F. V. Fomin, D. Lokshtanov, S. Saurabh, and Y. Villanger. “Kernel(s) for problems with no kernel: on out-trees with many leaves”. In: *ACM Transactions on Algorithms* 8.4 (2012), 38:1–38:19. URL: https://doi.org/10.1145/2344422.2344428 (cit. on p. 26).

[Blä03] M. Bläser. “Computing small partial coverings”. In: *Information Processing Letters* 85.6 (2003), pp. 327–331. URL: https://doi.org/10.1016/S0020-0190(02)00434-9 (cit. on p. 13).

[BLP02] J. Bourjolly, G. Laporte, and G. Pesant. “An exact algorithm for the maximum k-club problem in an undirected graph”. In: *European Journal of Operational Research* 138.1 (2002), pp. 21–28. URL: https://doi.org/10.1016/S0377-2217(01)00133-3 (cit. on pp. 15, 16, 38, 61, 69).

[BMN12] R. van Bevern, H. Moser, and R. Niedermeier. “Approximation and tidying - A problem kernel for s-plex cluster vertex deletion”. In: *Algorithmica* 62.3-4 (2012), pp. 930–950. URL: https://doi.org/10.1007/s00453-011-9492-7 (cit. on p. 15).

[BN19] M. Bentert and A. Nichterlein. “Parameterized complexity of diameter”. In: *Algorithms and Complexity - 11th International Conference, CIAC 2019*. Vol. 11485. Lecture Notes in Computer Science. Springer, 2019, pp. 50–61. URL: https://doi.org/10.1007/978-3-030-17402-6_5 (cit. on p. 87).

[Bog+14] V. Boginski, S. Butenko, O. Shirokikh, S. Trukhanov, and J. Gil-Lafuente. “A network-based data mining approach to portfolio selection via weighted clique relaxations”. In: *Annals of Operations Research* 216.1 (2014), pp. 23–34. URL: https://doi.org/10.1007/s10479-013-1395-3 (cit. on p. 9).

[Buc+14] A. Buchanan, J. L. Walteros, S. Butenko, and P. M. Pardalos. “Solving maximum clique in sparse graphs: an $\mathcal{O}(nm+n2^{d/4})$ algorithm for d-degenerate graphs”. In: *Optimization Letters* 8.5 (2014), pp. 1611–1617. URL: https://doi.org/10.1007/s11590-013-0698-2 (cit. on pp. 9, 16).

[CA11] F. D. Carvalho and M. T. Almeida. “Upper bounds and heuristics for the 2-club problem”. In: *European Journal of Operational Research* 210.3 (2011), pp. 489–494. URL: http://doi.org/10.1016/j.ejor.2010.11.023 (cit. on p. 16).

[Cai08] L. Cai. “Parameterized complexity of cardinality constrained optimization problems”. In: *The Computer Journal* 51.1 (2008), pp. 102–121. URL: https://doi.org/10.1093/comjnl/bxm086 (cit. on pp. 13, 19, 39, 76).

[Cai+97] L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. “Advice classes of parameterized tractability”. In: *Annals of Pure and Applied Logic* 84.1 (1997), pp. 119–138. URL: https://doi.org/10.1016/S0168-0072(95)00020-8 (cit. on p. 26).

[Cas+14] B. Caskurlu, V. Mkrtchyan, O. Parekh, and K. Subramani. “On partial vertex cover and budgeted maximum coverage problems in bipartite graphs”. In: *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014*. Vol. 8705. Lecture Notes in Computer Science. Springer, 2014, pp. 13–26. URL: https://doi.org/10.1007/978-3-662-44602-7_2 (cit. on p. 13).

[CFG05] Y. Chen, J. Flum, and M. Grohe. “Machine-based methods in parameterized complexity theory”. In: *Theoretical Computer Science* 339.2-3 (2005), pp. 167–199. URL: https://doi.org/10.1016/j.tcs.2005.02.003 (cit. on p. 33).

[Cha+13] M. Chang, L. Hung, C. Lin, and P. Su. “Finding large k-clubs in undirected graphs”. In: *Computing* 95.9 (2013), pp. 739–758. URL: https://doi.org/10.1007/s00607-012-0263-3 (cit. on pp. 15, 16).

[Che+20] P. Chen, H. Wan, S. Cai, J. Li, and H. Chen. “Local search with dynamic-threshold configuration checking and incremental neighborhood updating for maximum k-plex problem”. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, USA, 2020*. AAAI Press, 2020, pp. 2343–2350. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5613 (cit. on p. 15).

[Che+21] X. Chen, Y. Zhou, J.-K. Hao, and M. Xiao. “Computing maximum k-defective cliques in massive graphs”. In: *Computers & Operations Research* 127 (2021 - in press), pp. 105–131. URL: http://www.sciencedirect.com/science/article/pii/S0305054820302483 (cit. on pp. 9, 13, 80, 105).

[CKJ01] J. Chen, I. A. Kanj, and W. Jia. “Vertex cover: further observations and further improvements”. In: *Journal of Algorithms* 41.2 (2001), pp. 280–301. URL: https://doi.org/10.1006/jagm.2001.1186 (cit. on pp. 17, 19, 44).

[Coh08] J. Cohen. *Trusses: Cohesive subgraphs for social network analysis*. National security agency technical report 3.1. 2008 (cit. on p. 10).

[Con+17] A. Conte, D. Firmani, C. Mordente, M. Patrignani, and R. Torlone. “Fast enumeration of large k-plexes”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Canada, 2017*. ACM, 2017, pp. 115–124. URL: https://doi.org/10.1145/3097983.3098031 (cit. on p. 15).

[CP90] R. Carraghan and P. M. Pardalos. “An exact algorithm for the maximum clique problem”. In: *Operations Research Letters* 9.6 (1990), pp. 375 – 382. URL: http://www.sciencedirect.com/science/article/pii/016763779090057C (cit. on pp. 9, 10).

[Cyg+15] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015. URL: https://doi.org/10.1007/978-3-319-21275-3 (cit. on pp. 25, 26, 43–45).

[DF13] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. URL: https://doi.org/10.1007/978-1-4471-5559-1 (cit. on p. 25).

[Die16] R. Diestel. *Graph Theory, 5th Edition*. Vol. 173. Graduate Texts in Mathematics. Springer, 2016 (cit. on pp. 21, 31).

[DIM12] DIMACS’12. *Graph partitioning and graph clustering. 10th DIMACS implementation challenge*. https://www.cc.gatech.edu/dimacs10/downloads.shtml. Accessed March 2021. 2012 (cit. on pp. 63, 103).

[DIM93] DIMACS’93. *NP-hard problems: Maximum clique, graph coloring, and satisfiability. 2nd DIMACS implementation challenge*. https://dmac.rutgers.edu/pub/challenge/. Accessed March 2021. 1993 (cit. on pp. 63, 103).

[EH66] P. Erdős and A. Hajnal. “On chromatic number of graphs and set-systems”. In: *Acta Mathematica Academiae Scientiarum Hungarica* 17.1-2 (1966), pp. 61–99 (cit. on p. 16).

[EL05] E. Elmacioglu and D. Lee. “On six degrees of separation in DBLP-DB and more”. In: *ACM SIGMOD Record* 34.2 (2005), pp. 33–40. URL: https://doi.org/10.1145/1083784.1083791 (cit. on p. 12).

[Est+05] V. Estivill-Castro, M. R. Fellows, M. A. Langston, and F. A. Rosamond. “FPT is P-time extremal structure I”. In: *Algorithms and Complexity in Durham 2005 - Proceedings of the First ACiD Workshop, Durham, UK*. Vol. 4. Texts in Algorithmics. King’s College, London, 2005, pp. 1–41 (cit. on p. 26).

[Fel+11] M. R. Fellows, J. Guo, H. Moser, and R. Niedermeier. “A generalization of Nemhauser and Trotter’s local optimization theorem”. In: *Journal of Computer and System Sciences* 77.6 (2011), pp. 1141 –1158. URL: http://www.sciencedirect.com/science/article/pii/S002200001000142X (cit. on pp. 14, 17, 19, 38, 39, 55, 82).

[FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. URL: https://doi.org/10.1007/3-540-29953-X (cit. on pp. 25, 26).

[Fig+21] A. Figiel, A. Himmel, A. Nichterlein, and R. Niedermeier. “On 2-clubs in graph-based data clustering: theory and algorithm engineering”. In: *Proceedings of the 11th International Conference on Algorithms and Complexity (CIAC ’21)*. Lecture Notes in Computer Science. Accepted for publication. Springer, 2021. URL: https://arxiv.org/abs/2006.14972 (cit. on p. 16).

[Fom+18] F. Fomin, D. Lokshtanov, S. Saurabh, and M. Zehavi. *Kernelization: Theory of Parameterized Preprocessing.* Cambridge University Press, Dec. 2018 (cit. on p. 26).

[FS97] U. Feige and M. Seltser. *On the Densest k-Subgraph Problems.* Technical Report. ISR, 1997 (cit. on pp. 23, 27, 30, 32).

[FT14] M. Farach-Colton and M. Tsai. "Computing the degeneracy of large graphs". In: *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, 2014.* Vol. 8392. Lecture Notes in Computer Science. Springer, 2014, pp. 250–260. URL: `https://doi.org/10.1007/978-3-642-54423-1_22` (cit. on p. 90).

[FT16] M. Farach-Colton and M. Tsai. "Tight approximations of degeneracy in large graphs". In: *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, 2016.* Vol. 9644. Lecture Notes in Computer Science. Springer, 2016, pp. 429–440. URL: `https://doi.org/10.1007/978-3-662-49529-2_32` (cit. on p. 90).

[GIP18] T. Gschwind, S. Irnich, and I. Podlinski. "Maximum weight relaxed cliques and russian doll search revisited". In: *Discrete Applied Mathematics* 234 (2018), pp. 131–138. URL: `https://doi.org/10.1016/j.dam.2016.09.039` (cit. on pp. 9, 13, 15).

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979 (cit. on pp. 23, 27, 28, 70).

[GNW07] J. Guo, R. Niedermeier, and S. Wernicke. "Parameterized complexity of vertex cover variants". In: *Theory of Computing Systems* 41.3 (2007), pp. 501–520. URL: `https://doi.org/10.1007/s00224-007-1309-3` (cit. on pp. 13, 19, 28, 33, 39, 76).

[Gsc+17] T. Gschwind, S. Irnich, F. Furini, and R. W. Calvo. *Social Network Analysis and Community Detection by Decomposing a Graph into Relaxed Cliques.* Working Papers 1722. Gutenberg School of Management and Economics, Johannes Gutenberg-Universität Mainz, Dec. 2017. URL: `https://ideas.repec.org/p/jgu/wpaper/1722.html` (cit. on pp. 12, 13, 15, 16).

[Gsc+20] T. Gschwind, S. Irnich, F. Furini, and R. W. Calvo. "A branch-and-price framework for decomposing graphs into relaxed cliques". In: *INFORMS Journal on Computing* (2020 - in press) (cit. on pp. 10, 12–16, 77).

[Guo+11] J. Guo, I. A. Kanj, C. Komusiewicz, and J. Uhlmann. "Editing graphs into disjoint unions of dense clusters". In: *Algorithmica* 61.4 (2011), pp. 949–970. URL: `https://doi.org/10.1007/s00453-011-9487-4` (cit. on pp. 10, 13, 15, 23, 27, 40).

[Har+15] S. Hartung, C. Komusiewicz, A. Nichterlein, and O. Suchý. "On structural parameterizations for the 2-club problem". In: *Discrete Applied Mathematics* 185 (2015), pp. 79–92. URL: `https://doi.org/10.1016/j.dam.2014.11.026` (cit. on pp. 16, 19, 35, 39, 89, 105).

[HK73] J. E. Hopcroft and R. M. Karp. “An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs”. In: *SIAM Journal on Computing* 2.4 (1973), pp. 225–231. URL: https://doi.org/10.1137/0202019 (cit. on p. 44).

[HKL19] S. Hsieh, S. Kao, and Y. Lin. “A swap-based heuristic algorithm for the maximum k-plex problem”. In: *IEEE Access* 7 (2019), pp. 110267–110278. URL: https://doi.org/10.1109/ACCESS.2019.2934470 (cit. on p. 15).

[HKN15a] S. Hartung, C. Komusiewicz, and A. Nichterlein. “Parameterized algorithmics and computational experiments for finding 2-clubs”. In: *Journal of Graph Algorithms and Applications* 19.1 (2015), pp. 155–190. URL: https://doi.org/10.7155/jgaa.00352 (cit. on pp. 9, 10, 15, 16, 87, 90, 103–105).

[HKN15b] S. Hartung, C. Komusiewicz, and A. Nichterlein. *graph-thres instances*. https://fpt.akt.tu-berlin.de/two-club-data/coauthor/. Accessed February 2021. 2015 (cit. on pp. 63, 103).

[HS00] E. Hartuv and R. Shamir. “A clustering algorithm based on graph connectivity”. In: *Information Processing Letters* 76.4-6 (2000), pp. 175–181. URL: https://doi.org/10.1016/S0020-0190(00)00142-3 (cit. on p. 10).

[IPZ01] R. Impagliazzo, R. Paturi, and F. Zane. “Which problems have strongly exponential complexity?” In: *Journal of Computer and System Sciences* 63.4 (2001), pp. 512–530. URL: https://doi.org/10.1006/jcss.2001.1774 (cit. on pp. 16, 17, 19, 64, 90).

[JP09] D. Jiang and J. Pei. “Mining frequent cross-graph quasi-cliques”. In: *ACM Transactions on Knowledge Discovery from Data* 2.4 (2009), 16:1–16:42. URL: https://doi.org/10.1145/1460797.1460799 (cit. on p. 10).

[Kad68] C. Kadushin. “Power, influence and social circles: a new methodology for studying opinion makers”. In: *American Sociological Review* 33.5 (1968), pp. 685–699. URL: http://www.jstor.org/stable/2092880 (cit. on p. 9).

[Kar72] R. M. Karp. “Reducibility among combinatorial problems”. In: *Proceedings of a Symposium on the Complexity of Computer Computations, IBM 1972.* The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103. URL: https://doi.org/10.1007/978-1-4684-2001-2_9 (cit. on pp. 9, 13, 39).

[Kha80] L. Khachiyan. “Polynomial algorithms in linear programming”. In: *USSR Computational Mathematics and Mathematical Physics* 20.1 (1980), pp. 53–72. URL: http://www.sciencedirect.com/science/article/pii/0041555380900610 (cit. on p. 43).

[KKS20] T. Koana, C. Komusiewicz, and F. Sommer. “Exploiting c-closure in kernelization algorithms for graph problems”. In: *28th Annual European Symposium on Algorithms, ESA 2020.* Vol. 173. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 65:1–65:17. URL: https://doi.org/10.4230/LIPIcs.ESA.2020.65 (cit. on pp. 13–15, 19, 30, 35, 83, 89).

[Kom+09] C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. “Isolation concepts for efficiently enumerating dense subgraphs”. In: *Theoretical Computer Science* 410.38-40 (2009), pp. 3640–3654. URL: https://doi.org/10.1016/j.tcs.2009.04.021 (cit. on pp. 14, 15, 19, 81, 82, 89).

[Kom11] C. Komusiewicz. “Parameterized Algorithmics for Network Analysis: Clustering & Querying”. PhD thesis. Technical University of Berlin, 2011. URL: http://opus.kobv.de/tuberlin/volltexte/2011/3250/ (cit. on pp. 5, 6, 10, 11, 19, 29, 37, 57, 70, 89).

[Kom16] C. Komusiewicz. “Multivariate algorithmics for finding cohesive subnetworks”. In: *Algorithms* 9.1 (2016), p. 21. URL: https://doi.org/10.3390/a9010021 (cit. on pp. 10, 12–15, 57, 66, 68).

[Kom+19] C. Komusiewicz, A. Nichterlein, R. Niedermeier, and M. Picker. “Exact algorithms for finding well-connected 2-clubs in sparse real-world graphs: theory and experiments”. In: *European Journal of Operational Research* 275.3 (2019), pp. 846–864. URL: https://doi.org/10.1016/j.ejor.2018.12.006 (cit. on pp. 16, 58, 64, 68, 69, 104, 105).

[Kos04] S. Kosub. “Local density”. In: *Network Analysis: Methodological Foundations [outcome of a Dagstuhl seminar, 2004]*. Vol. 3418. Lecture Notes in Computer Science. Springer, 2004, pp. 112–142. URL: https://doi.org/10.1007/BF02786825 (cit. on pp. 14, 77, 85, 86).

[KP93] G. Kortsarz and D. Peleg. “On choosing a dense subgraph (extended abstract)”. In: *34th Annual Symposium on Foundations of Computer Science, USA, 1993*. IEEE Computer Society, 1993, pp. 692–701. URL: https://doi.org/10.1109/SFCS.1993.366818 (cit. on pp. 10, 23, 27).

[KS15] C. Komusiewicz and M. Sorge. “An algorithmic framework for fixed-cardinality optimization in sparse graphs applied to dense subgraph problems”. In: *Discrete Applied Mathematics* 193 (2015), pp. 145–161. URL: https://doi.org/10.1016/j.dam.2015.04.029 (cit. on pp. 14, 30, 31).

[KWB05] A. M. C. A. Koster, T. Wolle, and H. L. Bodlaender. “Degree-based treewidth lower bounds”. In: *Experimental and Efficient Algorithms, 4th International Workshop, WEA 2005, Greece*. Vol. 3503. Lecture Notes in Computer Science. Springer, 2005, pp. 101–112. URL: https://doi.org/10.1007/11427186_11 (cit. on p. 16).

[LK14] J. Leskovec and A. Krevl. *SNAP Datasets: Stanford large network dataset collection*. http://snap.stanford.edu/data. Accessed March 2021. June 2014 (cit. on pp. 63, 103).

[LP49] R. D. Luce and A. D. Perry. “A method of matrix analysis of group structure”. In: *Psychometrika* 14.2 (1949), pp. 95–116 (cit. on pp. 23, 27).

[Luc50] R. D. Luce. “Connectivity and generalized cliques in sociometric group structure”. In: *Psychometrika* 15.2 (1950), pp. 169–190 (cit. on p. 10).

[LW70] D. R. Lick and A. T. White. “k-degenerate graphs”. In: *Canadian Journal of Mathematics* 22.5 (1970), 1082–1096 (cit. on pp. 9, 10, 16).

[LY80] J. M. Lewis and M. Yannakakis. "The node-deletion problem for hereditary properties is NP-complete". In: *Journal of Computer and System Sciences* 20.2 (1980), pp. 219–230. URL: https://doi.org/10.1016/0022-0000(80)90060-4 (cit. on pp. 13, 14, 66).

[LZZ12] H. Liu, P. Zhang, and D. Zhu. "On editing graphs into 2-club clusters". In: *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management - Joint International Conference, FAW-AAIM 2012*. Vol. 7285. Lecture Notes in Computer Science. Springer, 2012, pp. 235–246. URL: https://doi.org/10.1007/978-3-642-29700-7_22 (cit. on p. 16).

[Mad78] W. Mader. "Über n-fach zusammenhängende Eckenmengen in Graphen". In: *Journal of Combinatorial Theory, Series B* 25.1 (1978), pp. 74–93. URL: https://doi.org/10.1016/S0095-8956(78)80012-4 (cit. on p. 10).

[MB83] D. W. Matula and L. L. Beck. "Smallest-last ordering and clustering and graph coloring algorithms". In: *Journal of the ACM* 30.3 (1983), pp. 417–427. URL: https://doi.org/10.1145/2402.322385 (cit. on pp. 16, 19, 59, 64).

[Meh84] K. Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*. Vol. 2. EATCS Monographs on Theoretical Computer Science. Springer, 1984. URL: https://doi.org/10.1007/978-3-642-69897-2 (cit. on pp. 19, 40).

[MH12] B. McClosky and I. V. Hicks. "Combinatorial algorithms for the maximum k-plex problem". In: *Journal of Combinatorial Optimization* 23.1 (2012), pp. 29–49. URL: https://doi.org/10.1007/s10878-010-9338-2 (cit. on p. 15).

[MIH99] H. Matsuda, T. Ishihara, and A. Hashimoto. "Classifying molecular sequences using a linkage graph with their pairwise similarities". In: *Theoretical Computer Science* 210.2 (1999), pp. 305–325. URL: https://www.sciencedirect.com/science/article/pii/S0304397598000917 (cit. on p. 10).

[Mil67] S. Milgram. "The small world problem". In: *Psychology today* 2.1 (1967), pp. 60–67 (cit. on p. 12).

[MK13] E. Moreno-Centeno and R. M. Karp. "The implicit hitting set approach to solve combinatorial optimization problems with an application to multi-genome alignment". In: *Operations Research* 61.2 (2013), pp. 453–468. URL: https://doi.org/10.1287/opre.1120.1139 (cit. on p. 70).

[MNS12] H. Moser, R. Niedermeier, and M. Sorge. "Exact combinatorial algorithms and experiments for finding maximum k-plexes". In: *Journal of Combinatorial Optimization* 24.3 (2012), pp. 347–373. URL: https://doi.org/10.1007/s10878-011-9391-5 (cit. on pp. 14, 15, 19, 39, 42, 55, 82).

[Mok79] R. J. Mokken. "Cliques, clubs and clans". In: *Quality & Quantity* 13.2 (1979), pp. 161–173 (cit. on pp. 10, 15, 23, 27).

[MPS20] N. Misra, F. Panolan, and S. Saurabh. “Subexponential algorithm for d-cluster edge deletion: exception or rule?” In: *Journal of Computer and System Sciences* 113 (2020), pp. 150–162. URL: https://doi.org/10.1016/j.jcss.2020.05.008 (cit. on p. 16).

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006 (cit. on p. 25).

[NRT05] N. Nishimura, P. Ragde, and D. M. Thilikos. “Fast fixed-parameter tractable algorithms for nontrivial generalizations of vertex cover”. In: *Discrete Applied Mathematics* 152.1-3 (2005), pp. 229–245. URL: https://doi.org/10.1016/j.dam.2005.02.029 (cit. on pp. 14, 38).

[NT75] G. L. Nemhauser and L. E. Trotter. “Vertex packings: structural properties and algorithms”. In: *Mathematical Programming* 8.1 (1975), pp. 232–248. URL: https://doi.org/10.1007/BF01580444 (cit. on pp. 17, 19, 44).

[Öst02] P. R. J. Östergård. “A fast algorithm for the maximum clique problem”. In: *Discrete Applied Mathematics* 120.1-3 (2002), pp. 197–207. URL: https://doi.org/10.1016/S0166-218X(01)00290-6 (cit. on p. 9).

[PB12] F. M. Pajouh and B. Balasundaram. “On inclusionwise maximal and maximum cardinality k-clubs in graphs”. In: *Discrete Optimization* 9.2 (2012), pp. 84–97. URL: https://doi.org/10.1016/j.disopt.2012.02.002 (cit. on p. 16).

[Pic15] M. Picker. *Algorithms and experiments for finding robust 2-clubs*. Master Thesis. 2015. URL: https://fpt.akt.tu-berlin.de/publications/theses/MA-marten-picker.pdf (cit. on pp. 16, 59).

[PR05] E. Prieto-Rodríguez. “Systematic kernelization in FPT algorithm design”. PhD thesis. University of Newcastle, Australia, 2005 (cit. on pp. 43, 45, 46).

[Pul20] W. Pullan. “Local search for the maximum k-plex problem”. In: *Journal of Heuristics* (2020), pp. 1–22 (cit. on pp. 15, 105).

[PYB13] J. Pattillo, N. Youssef, and S. Butenko. “On clique relaxation models in network analysis”. In: *European Journal of Operational Research* 226.1 (2013), pp. 9–18. URL: https://doi.org/10.1016/j.ejor.2012.10.021 (cit. on pp. 10–12, 14, 66, 75, 77, 80, 81, 84–86, 90).

[RGG15] R. A. Rossi, D. F. Gleich, and A. H. Gebremedhin. “Parallel maximum clique algorithms with applications to network analysis”. In: *SIAM Journal on Scientific Computing* 37.5 (2015), pp. C589–C616. URL: https://doi.org/10.1137/14100018X (cit. on p. 9).

[RS08] V. Raman and S. Saurabh. “Short cycles make W-hard problems hard: FPT algorithms for W-hard problems in graphs with no short cycles”. In: *Algorithmica* 52.2 (2008), pp. 203–225. URL: https://doi.org/10.1007/s00453-007-9148-9 (cit. on pp. 10, 13, 14, 19, 23, 28, 30, 31, 39, 40, 80, 81, 89).

[RW13] L. Roditty and V. V. Williams. “Fast approximation algorithms for the diameter and radius of sparse graphs”. In: *Symposium on Theory of Computing Conference, STOC’13*. ACM, 2013, pp. 515–524. URL: https://doi.org/10.1145/2488608.2488673 (cit. on p. 64).

[SB20] H. Salemi and A. Buchanan. “Parsimonious formulations for low-diameter clusters”. In: *Mathematical Programming Computation* 12.3 (2020), pp. 493–528. URL: https://doi.org/10.1007/s12532-020-00175-6 (cit. on pp. 16, 104, 105).

[Sch+12] A. Schäfer, C. Komusiewicz, H. Moser, and R. Niedermeier. “Parameterized computational complexity of finding small-diameter subgraphs”. In: *Optimization Letters* 6.5 (2012), pp. 883–891. URL: https://doi.org/10.1007/s11590-011-0311-5 (cit. on pp. 15, 16, 19, 38, 67, 69, 87, 89).

[SF78] S. B. Seidman and B. L. Foster. “A graph-theoretic generalization of the clique concept”. In: *Journal of Mathematical Sociology* 6.1 (1978), pp. 139–154 (cit. on pp. 10, 14, 15, 23, 27, 77, 82).

[SGB14] P. Sun, J. Guo, and J. Baumbach. “Complexity of dense bicluster editing problems”. In: *Computing and Combinatorics - 20th International Conference, COCOON 2014, USA. Proceedings.* Vol. 8591. Lecture Notes in Computer Science. Springer, 2014, pp. 154–165. URL: https://doi.org/10.1007/978-3-319-08783-2_14 (cit. on pp. 13, 15).

[Shi13] O. A. Shirokikh. “Degree-based Clique Relaxations: Theoretical Bounds, Computational Issues, and Applications”. PhD thesis. University of Florida, 2013 (cit. on pp. 9, 11, 13, 15, 39, 75, 77).

[SP15] K. K. Singh and A. K. Pandey. “Survey of algorithms on maximum clique problem”. In: *International Advanced Research Journal in Science, Engineering and Technology* 2.2 (2015), pp. 18–20 (cit. on p. 12).

[SS06] H. D. Sherali and J. C. Smith. “A polyhedral study of the generalized vertex packing problem”. In: *Mathematical Programming* 107.3 (2006), pp. 367–390. URL: https://doi.org/10.1007/s10107-004-0504-0 (cit. on p. 13).

[Sto+20] V. Stozhkov, A. Buchanan, S. Butenko, and V. Boginski. “Continuous cubic formulations for cluster detection problems in networks”. In: *Mathematical Programming* 12 (2020), pp. 1–29 (cit. on pp. 9, 13, 15).

[SUS07] R. Sharan, I. Ulitsky, and R. Shamir. “Network-based prediction of protein function”. In: *Molecular Systems Biology* 3.1 (2007), p. 88. eprint: https://www.embopress.org/doi/pdf/10.1038/msb4100129. URL: https://www.embopress.org/doi/abs/10.1038/msb4100129 (cit. on p. 9).

[SW19] M. Sorge and M. Weller. *The graph parameter hierarchy.* Unpublished Manuscript. 2019. URL: https://manyu.pro/assets/parameter-hierarchy.pdf (cit. on p. 62).

[Tho10] S. Thomassé. “A $4k^2$ kernel for feedback vertex set”. In: *ACM Transactions on Algorithms* 6.2 (Apr. 2010). URL: https://doi.org/10.1145/1721837.1721848 (cit. on p. 47).

[Tru+13] S. Trukhanov, C. Balasubramaniam, B. Balasundaram, and S. Butenko. "Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations". In: *Computational Optimization and Applications* 56.1 (2013), pp. 113–130. URL: https://doi.org/10.1007/s10589-013-9548-5 (cit. on pp. 9–11, 13, 15–17, 39, 59, 75, 83).

[VB12] A. Veremyev and V. Boginski. "Identifying large robust network clusters via new compact formulations of maximum k-club problems". In: *European Journal of Operational Research* 218.2 (2012), pp. 316–326. URL: https://doi.org/10.1016/j.ejor.2011.10.027 (cit. on p. 16).

[VBB15] A. Verma, A. Buchanan, and S. Butenko. "Solving the maximum clique and vertex coloring problems on very large sparse networks". In: *INFORMS Journal on Computing* 27.1 (2015), pp. 164–177. URL: https://doi.org/10.1287/ijoc.2014.0618 (cit. on p. 10).

[Ver+14] A. Veremyev, O. A. Prokopyev, V. Boginski, and E. L. Pasiliao. "Finding maximum subgraphs with relatively large vertex connectivity". In: *European Journal of Operational Research* 239.2 (2014), pp. 349–362. URL: https://doi.org/10.1016/j.ejor.2014.05.041 (cit. on p. 10).

[VLS96] G. Verfaillie, M. Lemaître, and T. Schiex. "Russian doll search for solving constraint optimization problems". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, USA, Volume 1*. AAAI Press / The MIT Press, 1996, pp. 181–187. URL: http://www.aaai.org/Library/AAAI/1996/aaai96-027.php (cit. on p. 9).

[WB20] J. L. Walteros and A. Buchanan. "Why is maximum clique often easy in practice?" In: *Operations Research* 68.6 (2020), pp. 1866–1895. URL: https://doi.org/10.1287/opre.2019.1970 (cit. on pp. 5, 6, 9–12, 19, 24, 29, 37, 57, 67, 70, 88–90, 104).

[WF94] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994. URL: https://doi.org/10.1017/CBO9780511815478 (cit. on p. 11).

[WH15] Q. Wu and J. Hao. "A review on algorithms for maximum clique problems". In: *European Journal of Operational Research* 242.3 (2015), pp. 693–709. URL: https://doi.org/10.1016/j.ejor.2014.09.064 (cit. on p. 12).

[Xia+17] M. Xiao, W. Lin, Y. Dai, and Y. Zeng. "A fast algorithm to compute maximum k-plexes in social network analysis". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, 2017, USA*. AAAI Press, 2017, pp. 919–925. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14500 (cit. on pp. 14, 15, 85).

[YPB17] O. Yezerska, F. M. Pajouh, and S. Butenko. "On biconnected and fragile subgraphs of low diameter". In: *European Journal of Operational Research* 263.2 (2017), pp. 390–400. URL: https://doi.org/10.1016/j.ejor.2017.05.020 (cit. on p. 16).

[Yu+06] H. Yu, A. Paccanaro, V. Trifonov, and M. Gerstein. “Predicting interactions in protein networks by completing defective cliques”. In: *Bioinformatics* 22.7 (2006), pp. 823–829. URL: `https://doi.org/10.1093/bioinformatics/btl014` (cit. on pp. 9–11, 13, 23, 27).

[Zak13] M. Zaker. “Generalized degeneracy, dynamic monopolies and maximum degenerate subgraphs”. In: *Discrete Applied Mathematics* 161.16-17 (2013), pp. 2716–2723. URL: `https://doi.org/10.1016/j.dam.2013.04.012` (cit. on p. 16).

[ZH17] Y. Zhou and J. Hao. “Frequency-driven tabu search for the maximum s-plex problem”. In: *Computers & Operations Research* 86 (2017), pp. 65–78. URL: `https://doi.org/10.1016/j.cor.2017.05.005` (cit. on pp. 15, 105).

Appendix A

Appendix - Data Tables

We provide the values of the respective "relaxed gaps" of Maximum Connected s-Defective Clique, Maximum Connected s-Plex, and Maximum s-Club, thus complementing the algorithms of Chapter 6. In order to compute these relaxed gaps, first we computed the weak x-degeneracy with $x \in [1, 4]$ according to Definition 5.1 for many real-world graphs, the results are presented in Tables A.1 to A.4. We also computed the strong x-degeneracy according to Definition 5.3 along the way, as we want to present how the weak and strong x-degeneracies relate in real-world graphs. Recall from Observation 5.4 that the strong x-degeneracy of a graph G is the standard degeneracy of its power graph G^x. Afterwards, in Tables A.5 to A.8, we provide an overview of the relaxed gaps for the three Maximum Clique relaxations for $s \in [1, 4]$. Our goal is to provide more information about how we computed and acquired the values in these tables. The code to compute the generalized degeneracies as well as a large CSV-file with all values from the Tables A.1 to A.8 is made publicly available under https://git.tu-berlin.de/niklas.wuensche/compute-diameter-and-weak-and-strong-degeneracies/.

Computing generalized degeneracies. In Tables A.1 to A.4, we computed the weak/strong x-degeneracies for $x \in [1, 4]$ of 151 graphs from the 2nd and 10th DIMACS challenges [DIM12; DIM93] and the SNAP repository [LK14]. Furthermore, we studied the `graph-thres`-instances [HKN15b], which were first presented and motivated by Hartung, Komusiewicz, and Nichterlein [HKN15a]. We renamed the graphs `kron_g500-simple-logn16` and `preferentialAttachment` to `kron500-16` and `prefAtt`.

We ran our program on an Intel(R) Xeon(R) W-2125 CPU @ 4.00GHz with four cores and eight threads and with 220GB main memory under the Ubuntu 18.04.5 operating system. Our implementation is written in C++ and was compiled with g++ version 7.5.0. The program depends on the libboost-dev library in version 1.65.1.0ubuntu1, and the CTPL library in version 0.0.2.

Recall that the x-neighborhood of a vertex v in a graph G contains all vertices with distance at most x to v in G, except v itself, and G^x is the graph where each vertex is adjacent to all vertices in its x-neighborhood with respect to G. We implemented our algorithms for computing the weak x-degeneracy α_x of a graph G (Successively find and remove the vertex with the smallest x-neighborhood) in $\mathcal{O}(\alpha_x nm)$ time as described in Lemma 5.6, as well as our algorithm for computing the strong x-degeneracy β_x (compute the standard degeneracy of the power graph G^x) in $\mathcal{O}(nm)$ time as described in Lemma 5.7. We store the graphs as adjacency lists to reduce their sizes in memory.

Note that both algorithms could be implemented by storing the x-neighborhood of each vertex at all times. Hence, both algorithms would use $\mathcal{O}(\Delta^x n)$ space, where Δ is the maximum degree of the graph. However, this takes up too much space in practice, and we ran out of memory for a couple of graphs like the `eu-2005` graph. To solve this issue, we only stored the *size* of the x-neighborhoods, and re-computed the actual set when needed. This method only requires $\mathcal{O}(n)$ space at the cost of an $\mathcal{O}(nm)$-time overhead. Furthermore, note that both algorithms for computing the weak/strong x-degeneracies of a graph heavily rely on computing the x-neighborhoods of the vertices, which can be computed independently from each other in parallel. As we used a multi-core machine to compute all values, we implemented this parallelization.

For computing the strong x-degeneracy of a graph, we set a time limit of 24 hours. In contrast, as it took much longer to compute the weak x-degeneracy than the strong x-degeneracy of a graph in general, we had to set a time limit of six hours for computing the weak x-degeneracy of a graph. The computation time includes the time to read the graph from the file, and we rounded up the times to the next full second. If the time limit was reached, then in Tables A.1 to A.4 we set the respective running time in the table to "-", and provide the size of the largest x-neighborhood of a vertex which was already removed from G as a lower bound on the weak x-degeneracy. Note that the respective strong x-degeneracy is an upper-bound on the weak x-degeneracy due to Proposition 5.5. For the sake of completeness, we also computed the weak/strong 1-degeneracies of the graphs, although the standard degeneracy, the weak 1-degeneracy, and the strong 1-degeneracy of a graph are all by definition equivalent. We also computed the diameter and the maximum degree Δ of the graphs for comparison.

Computing gaps. In Tables A.5 to A.8, we computed the "gaps" for Maximum Connected s-Defective Clique, Maximum Connected s-Plex, and Maximum s-Club for $s \in [1, 4]$. We do not study Maximum Connected 0-defective Clique nor Maximum Connected 1-Plex nor Maximum 1-Club, because all these cases are equivalent to Maximum Clique, for which Walteros and Buchanan [WB20] already provided a detailed study on the size of the clique-core gap in real-world graphs.

Recall from Chapter 5 that for Maximum Connected s-Defective Clique, the corresponding gap parameter $g_{\text{C-sDC}}$ is defined as $\alpha_{\lfloor\sqrt{2s+\frac{1}{4}}+\frac{1}{2}\rfloor} + 1 - \omega_{\text{C-sDC}}$ due to Lemma 6.4, where $\omega_{\text{C-sDC}}$ is the maximum-order of a connected s-defective clique in the input graph. Similarly, for Maximum Connected s-Plex and Maximum s-Club, the corresponding gap parameters $g_{\text{C-sP}}$ and g_{sC} are $\alpha_s + 1 - \omega_{\text{C-sP}}$ and $\alpha_s + 1 - \omega_{\text{sC}}$ due to Observations 6.12 and 6.18. Hence, first we have to solve the maximization problems before we can compute the respective gaps. As it was out of focus of this work to implement a solver for these problems, we took the maximum solutions sizes for all three problems from the literature.

For Maximum 2-Club, we used the values from Hartung, Komusiewicz, and Nichterlein [HKN15a, Table 6] and Komusiewicz et al. [Kom+19, Table 3] as they looked at many different real-world graphs. Furthermore, we consider the paper of Salemi and Buchanan [SB20, Table 6] as this is one of the few papers which solves Maximum 3-

Club and Maximum 4-Club on some real-world graphs. To show where a certain solution size is taken from, we added the “cite”-column to our tables. If at least one of the values of the respective graph is taken from Hartung, Komusiewicz, and Nichterlein [HKN15a], Komusiewicz et al. [Kom+19], or Salemi and Buchanan [SB20], then we add the letter `H`, `K`, or `S` to the “cite”-column, respectively.

To the best of our knowledge, solving Maximum Connected s-Defective Clique and Maximum Connected s-Plex in practice has not been considered before, but only the standard Maximum s-Defective Clique and Maximum s-Plex problems. To still find the maximum solution sizes for Maximum Connected s-Defective Clique and Maximum Connected s-Plex, recall from Observation 6.6 that if an s-defective clique is of order at least $s+2$, then it is necessarily connected. In other words, if $\omega_{\text{sDC}} \geq s+2$, then $\omega_{\text{C-sDC}} = \omega_{\text{sDC}}$. Similarly, if $\omega_{\text{sP}} \geq 2s-1$, then $\omega_{\text{C-sP}} = \omega_{\text{sP}}$ due to Observation 6.14.

For Maximum s-defective Clique, we used the values from Chen et al. [Che+21, Tables 1, 2, 4, 5] as they studied many different real-world graphs. Note that all solutions they provided are necessarily connected.

For Maximum s-Plex, we used the values from Zhou and Hao [ZH17, Tables 1, 2, A1, A2] as they looked at many different real-world graphs. Note that they did not always provide the order of the largest s-plex in the graph. Hence, we only provide the size of $g_{\text{C-sP}}$ of a graph if the given ω_{sP} is known to be optimal. Furthermore, not all solutions they provided are necessarily connected, for example the maximum-order of an 4-plex in the graph `ecology1` is six, which is strictly smaller than $2s-1=7$. In cases like these, we set the value of the respective gap in Tables A.5 to A.8 to “?”. We mention that we were not able to consider the graphs `rgg_n_2_21_s0`, `rgg_n_2_22_s0`, `rgg_n_2_23_s0`, and `uk-2002` from their paper due to an out-of-memory error while reading these graphs.

If one of these papers provided the maximum-order of a clique relaxation, then we also provide the time it took them to compute this value. If the paper studied multiple solvers, then we provide the minimum over all of the given running times. We rounded up the times to the next full second. Furthermore, if in Tables A.1 to A.4 we could only provide an lower bound on the weak x-degeneracy which we need to compute the respective gap, then we only provide an lower bound on the size of the gap. Note that in this case, it could happen that the lower bound on the gap is negative as happens for the graphs `coPapersCiteseer` and `coPapersDBLP`. However, note that in Table A.5, the *exact* Maximum 2-Club-gap $g_{2\text{C}}$ of the `celegansneutral` graph is negative. We think that the maximum-order of a 2-club in this graph is not 285 (as claimed by Hartung, Komusiewicz, and Nichterlein [HKN15a, Table 6]), because Salemi and Buchanan [SB20, Table 6] claim that the maximum-order of a *3*-club is 243. As a 2-club is also a 3-club, we think that the value from Hartung, Komusiewicz, and Nichterlein should be smaller than 243.

Lastly, we want to thank the authors Hartung et al. [Har+15], Komusiewicz et al. [Kom+19], Pullan [Pul20], Salemi and Buchanan [SB20], and Zhou and Hao [ZH17] again for providing us their results in some machine-readable format.

Table A.1: Results for computing the weak/strong x-degeneracies of real-world graphs (part 1). We use the following notation: n–number of vertices, m–number of edges, dia–diameter, Δ–maximum degree, $x \in [1, 4]$, α_x–weak x-degeneracy, β_x–strong x-degeneracy, t_{α_x}–time (in seconds) to compute weak x-degeneracy (set to "-" if timeout after 6 hours), t_{β_x}–time (in seconds) to compute strong x-degeneracy (set to "-" if timeout after 24 hours). See Appendix A on page 103 for more details.

Graph	n	m	dia	Δ	Δ^2	α_1	α_2	α_3	α_4	β_1	β_2	β_3	β_4	t_{α_1}	t_{α_2}	t_{α_3}	t_{α_4}	t_{β_1}	t_{β_2}	t_{β_3}	t_{β_4}
johnson8-2-4	28	210	2	15	225	15	27	27	27	15	27	27	27	1	1	1	1	1	1	1	1
karate	34	78	5	17	289	4	17	24	32	4	17	24	32	1	1	1	1	1	1	1	1
chesapeake	39	170	3	33	1,089	6	35	38	38	6	35	38	38	1	1	1	1	1	1	1	1
MANN_a9	45	918	2	41	1,681	40	44	44	44	40	44	44	44	1	1	1	1	1	1	1	1
dolphins	62	159	8	12	144	4	16	28	39	4	16	29	39	1	1	1	1	1	1	1	1
hamming6-2	64	1,824	2	57	3,249	57	63	63	63	57	63	63	63	1	1	1	1	1	1	1	1
hamming6-4	64	704	2	22	484	22	63	63	63	22	63	63	63	1	1	1	1	1	1	1	1
johnson8-4-4	70	1,855	2	53	2,809	53	69	69	69	53	69	69	69	1	1	1	1	1	1	1	1
lesmis	77	508	5	72	5,184	9	36	57	74	9	37	63	74	1	1	1	1	1	1	1	1
polbooks	105	441	7	25	625	6	28	52	69	6	29	53	69	1	1	1	1	1	1	1	1
adjnoun	112	425	5	49	2,401	6	49	87	106	6	49	87	106	1	1	1	1	1	1	1	1
football	115	613	4	12	144	8	37	97	114	8	39	97	114	1	1	1	1	1	1	1	1
johnson16-2-4	120	5,460	2	91	8,281	91	119	119	119	91	119	119	119	1	1	1	1	1	1	1	1
C125.9	125	6,963	2	119	14,161	102	124	124	124	102	124	124	124	1	1	1	1	1	1	1	1
keller4	171	9,435	2	124	15,376	102	170	170	170	102	170	170	170	1	1	1	1	1	1	1	1
jazz	198	2,742	6	100	10,000	29	109	174	191	29	109	174	191	1	1	1	1	1	1	1	1
brock200_1	200	14,834	2	165	27,225	134	199	199	199	134	199	199	199	1	1	1	1	1	1	1	1
brock200_2	200	9,876	2	114	12,996	84	199	199	199	84	199	199	199	1	1	1	1	1	1	1	1
brock200_3	200	12,048	2	134	17,956	105	199	199	199	105	199	199	199	1	1	1	1	1	1	1	1
brock200_4	200	13,089	2	147	21,609	117	199	199	199	117	199	199	199	1	1	1	1	1	1	1	1
c-fat200-1	200	1,534	18	17	289	14	24	34	44	14	24	34	44	1	1	1	1	1	1	1	1
c-fat200-2	200	3,235	9	34	1,156	32	54	76	98	32	54	76	98	1	1	1	1	1	1	1	1
c-fat200-5	200	8,473	3	86	7,396	83	141	199	199	83	141	199	199	1	1	1	1	1	1	1	1
san200_0.7_1	200	13,930	2	155	24,025	125	199	199	199	125	199	199	199	1	1	1	1	1	1	1	1
san200_0.7_2	200	13,930	2	164	26,896	122	199	199	199	122	199	199	199	1	1	1	1	1	1	1	1
san200_0.9_1	200	17,910	2	191	36,481	162	199	199	199	162	199	199	199	1	1	1	1	1	1	1	1
san200_0.9_2	200	17,910	2	188	35,344	169	199	199	199	169	199	199	199	1	1	1	1	1	1	1	1
san200_0.9_3	200	17,910	2	187	34,969	169	199	199	199	169	199	199	199	1	1	1	1	1	1	1	1
sanr200_0.7	200	13,868	2	161	25,921	124	199	199	199	124	199	199	199	1	1	1	1	1	1	1	1
sanr200_0.9	200	17,863	2	189	35,721	166	199	199	199	166	199	199	199	1	1	1	1	1	1	1	1
gen200_p0.9_44	200	17,910	2	190	36,100	167	199	199	199	167	199	199	199	1	1	1	1	1	1	1	1
gen200_p0.9_55	200	17,910	2	190	36,100	166	199	199	199	166	199	199	199	1	1	1	1	1	1	1	1
C250.9	250	27,984	2	236	55,696	210	249	249	249	210	249	249	249	1	1	1	1	1	1	1	1
hamming8-2	256	31,616	2	247	61,009	247	255	255	255	247	255	255	255	1	1	1	1	1	1	1	1
hamming8-4	256	20,864	2	163	26,569	163	255	255	255	163	255	255	255	1	1	1	1	1	1	1	1
celegansneural	297	4,296	4	268	71,824	14	151	264	296	58	167	266	296	1	1	1	1	1	1	1	1
p_hat300-1	300	10,933	3	132	17,424	49	298	299	299	49	298	299	299	1	1	1	1	1	1	1	1
p_hat300-2	300	21,928	2	229	52,441	98	299	299	299	98	299	299	299	1	1	1	1	1	1	1	1

Table A.2: Results for computing the weak/strong x-degeneracies of real-world graphs (part 2). We use the following notation: n–number of vertices, m–number of edges, dia–diameter, Δ–maximum degree, $x \in [1, 4]$, α_x–weak x-degeneracy, β_x–strong x-degeneracy, t_{α_x}–time (in seconds) to compute weak x-degeneracy (set to "-" if timeout after 6 hours), t_{β_x}–time (in seconds) to compute strong x-degeneracy (set to "-" if timeout after 24 hours). See Appendix A on page 103 for more details.

Graph	n	m	dia	Δ	Δ^2	α_1	α_2	α_3	α_4	β_1	β_2	β_3	β_4	t_{α_1}	t_{α_2}	t_{α_3}	t_{α_4}	t_{β_1}	t_{β_2}	t_{β_3}	t_{β_4}
p_hat300-3	300	33,390	2	267	71,289	180	299	299	299	180	299	299	299	1	1	1	1	1	1	1	1
MANN_a27	378	70,551	2	374	$1.4{\cdot}10^5$	364	377	377	377	364	377	377	377	1	1	1	1	1	1	1	1
brock400_1	400	59,723	2	320	$1 \cdot 10^5$	277	399	399	399	277	399	399	399	2	1	1	1	1	1	1	1
brock400_2	400	59,786	2	328	$1.1{\cdot}10^5$	278	399	399	399	278	399	399	399	2	1	1	1	1	1	1	1
brock400_3	400	59,681	2	322	$1 \cdot 10^5$	278	399	399	399	278	399	399	399	2	1	1	1	1	1	1	1
brock400_4	400	59,765	2	326	$1.1{\cdot}10^5$	277	399	399	399	277	399	399	399	2	1	1	1	1	1	1	1
san400_0.5_1	400	39,900	2	225	50,625	183	399	399	399	183	399	399	399	1	1	1	1	1	1	1	1
san400_0.7_1	400	55,860	2	301	90,601	261	399	399	399	261	399	399	399	2	1	1	1	1	1	1	1
san400_0.7_2	400	55,860	2	304	92,416	259	399	399	399	259	399	399	399	2	1	1	1	1	1	1	1
san400_0.7_3	400	55,860	2	307	94,249	253	399	399	399	253	399	399	399	2	1	1	1	1	1	1	1
san400_0.9_1	400	71,820	2	374	$1.4{\cdot}10^5$	344	399	399	399	344	399	399	399	2	1	1	1	1	1	1	1
sanr400_0.5	400	39,984	2	233	54,289	177	399	399	399	177	399	399	399	1	1	1	1	1	1	1	1
sanr400_0.7	400	55,869	2	310	96,100	258	399	399	399	258	399	399	399	2	1	1	1	1	1	1	1
gen400_p0.9_55	400	71,820	2	375	$1.4{\cdot}10^5$	336	399	399	399	336	399	399	399	2	1	1	1	1	1	1	1
gen400_p0.9_65	400	71,820	2	378	$1.4{\cdot}10^5$	336	399	399	399	336	399	399	399	2	1	1	1	1	1	1	1
gen400_p0.9_75	400	71,820	2	380	$1.4{\cdot}10^5$	336	399	399	399	336	399	399	399	2	1	1	1	1	1	1	1
celegans_metabolic	453	2,025	7	237	56,169	10	237	372	431	10	237	372	431	1	1	1	1	1	1	1	1
johnson32-2-4	496	$1.1{\cdot}10^5$	2	435	$1.9{\cdot}10^5$	435	495	495	495	435	495	495	495	2	2	2	2	1	1	1	1
c-fat500-1	500	4,459	40	20	400	17	29	41	53	17	29	41	53	1	1	1	1	1	1	1	1
c-fat500-2	500	9,139	20	38	1,444	35	59	83	107	35	59	83	107	1	1	1	1	1	1	1	1
c-fat500-5	500	23,191	8	95	9,025	92	154	216	278	92	154	216	278	1	1	1	1	1	1	1	1
c-fat500-10	500	46,627	4	188	35,344	185	310	436	499	185	310	436	499	1	1	1	1	1	1	1	1
p_hat500-1	500	31,569	2	204	41,616	86	499	499	499	86	499	499	499	1	1	1	1	1	1	1	1
p_hat500-2	500	62,946	2	389	$1.5{\cdot}10^5$	170	499	499	499	170	499	499	499	2	2	2	2	1	1	1	1
p_hat500-3	500	93,800	2	452	$2 \cdot 10^5$	303	499	499	499	303	499	499	499	4	2	2	2	1	1	1	1
DSJC500.5	500	62,624	2	286	81,796	225	499	499	499	225	499	499	499	2	2	2	2	1	1	1	1
C500.9	500	$1.1{\cdot}10^5$	2	468	$2.2{\cdot}10^5$	432	499	499	499	432	499	499	499	4	3	3	3	1	1	1	1
p_hat700-1	700	60,999	2	286	81,796	117	699	699	699	117	699	699	699	2	3	3	3	1	1	1	1
p_hat700-2	700	$1.2{\cdot}10^5$	2	539	$2.9{\cdot}10^5$	235	699	699	699	235	699	699	699	6	5	5	5	1	1	1	1
p_hat700-3	700	$1.8{\cdot}10^5$	2	627	$3.9{\cdot}10^5$	426	699	699	699	426	699	699	699	12	7	7	7	1	1	1	1
keller5	776	$2.3{\cdot}10^5$	2	638	$4.1{\cdot}10^5$	560	775	775	775	560	775	775	775	12	10	10	10	1	1	1	1
brock800_1	800	$2.1{\cdot}10^5$	2	560	$3.1{\cdot}10^5$	487	799	799	799	487	799	799	799	18	10	10	10	2	1	1	1
brock800_2	800	$2.1{\cdot}10^5$	2	566	$3.2{\cdot}10^5$	486	799	799	799	486	799	799	799	18	10	10	10	2	1	1	1
brock800_3	800	$2.1{\cdot}10^5$	2	558	$3.1{\cdot}10^5$	483	799	799	799	483	799	799	799	18	10	10	10	2	1	1	1
brock800_4	800	$2.1{\cdot}10^5$	2	565	$3.2{\cdot}10^5$	485	799	799	799	485	799	799	799	18	10	10	10	2	1	1	1
p_hat1000-1	1,000	$1.2{\cdot}10^5$	2	408	$1.7{\cdot}10^5$	163	999	999	999	163	999	999	999	4	11	11	11	1	1	1	1

Table A.3: Results for computing the weak/strong x-degeneracies of real-world graphs (part 3). We use the following notation: n–number of vertices, m–number of edges, dia–diameter, Δ–maximum degree, $x \in [1,4]$, α_x–weak x-degeneracy, β_x–strong x-degeneracy, t_{α_x}–time (in seconds) to compute weak x-degeneracy (set to "-" if timeout after 6 hours), t_{β_x}–time (in seconds) to compute strong x-degeneracy (set to "-" if timeout after 24 hours). See Appendix A on page 103 for more details.

Graph	n	m	dia	Δ	Δ^2	α_1	α_2	α_3	α_4	β_1	β_2	β_3	β_4	t_{α_1}	t_{α_2}	t_{α_3}	t_{α_4}	t_{β_1}	t_{β_2}	t_{β_3}	t_{β_4}
p_hat1000-2	1,000	$2.4{\cdot}10^5$	2	766	$5.9{\cdot}10^5$	327	999	999	999	327	999	999	999	23	20	20	20	2	1	1	1
p_hat1000-3	1,000	$3.7{\cdot}10^5$	2	895	$8 \cdot 10^5$	609	999	999	999	609	999	999	999	42	27	27	27	3	1	1	1
san1000	1,000	$2.5{\cdot}10^5$	2	550	$3 \cdot 10^5$	464	999	999	999	464	999	999	999	14	19	18	18	2	1	1	1
DSJC1000.5	1,000	$2.5{\cdot}10^5$	2	551	$3 \cdot 10^5$	459	999	999	999	459	999	999	999	23	18	18	18	2	1	1	1
hamming10-2	1,024	$5.2{\cdot}10^5$	2	1,013	$1 \cdot 10^6$	1013	1023	1023	1023	1013	1023	1023	1023	35	34	34	34	2	1	1	1
hamming10-4	1,024	$4.3{\cdot}10^5$	2	848	$7.2{\cdot}10^5$	848	1023	1023	1023	848	1023	1023	1023	41	30	30	30	2	1	1	1
MANN_a45	1,035	$5.3{\cdot}10^5$	2	1,031	$1.1{\cdot}10^6$	1012	1034	1034	1034	1012	1034	1034	1034	37	37	37	37	2	2	2	2
email	1,133	5,451	8	71	5,041	11	87	384	801	11	93	390	802	1	1	2	3	1	1	1	1
polblogs	1,490	16,715	8	351	$1.2{\cdot}10^5$	36	376	864	1139	36	377	864	1139	1	3	5	6	1	1	1	1
p_hat1500-1	1,500	$2.8{\cdot}10^5$	2	614	$3.8{\cdot}10^5$	252	1499	1499	1499	252	1499	1499	1499	19	50	50	50	2	1	1	1
p_hat1500-2	1,500	$5.7{\cdot}10^5$	2	1,153	$1.3{\cdot}10^6$	504	1499	1499	1499	504	1499	1499	1499	107	94	94	94	5	2	2	2
p_hat1500-3	1,500	$8.5{\cdot}10^5$	2	1,330	$1.8{\cdot}10^6$	929	1499	1499	1499	929	1499	1499	1499	203	123	123	123	7	3	3	3
netscience	1,589	2,742	17	34	1,156	19	34	53	84	19	34	53	84	1	1	1	1	1	1	1	1
add20	2,395	7,462	15	123	15,129	35	123	670	1453	35	123	670	1453	1	1	4	11	1	1	1	1
data	2,851	15,093	79	17	289	7	23	45	73	7	23	45	73	1	1	1	1	1	1	1	1
MANN_a81	3,321	$5.5{\cdot}10^6$	2	3,317	$1.1{\cdot}10^7$	3280	3320	3320	3320	3280	3320	3320	3320	3941	3897	3921	3922	30	31	30	29
keller6	3,361	$4.6{\cdot}10^6$	2	2,952	$8.7{\cdot}10^6$	2690	3360	3360	3360	2690	3360	3360	3360	3121	3359	3371	3356	30	26	26	26
uk	4,824	6,837	214	3	9	2	6	11	17	2	6	11	17	1	1	1	1	1	1	1	1
power	4,941	6,594	46	19	361	5	19	29	60	5	19	29	60	1	1	1	1	1	1	1	1
add32	4,960	9,462	28	31	961	3	31	98	267	3	31	98	267	1	1	1	2	1	1	1	1
wiki-Vote	7,115	$1 \cdot 10^5$	7	1,065	$1.1{\cdot}10^6$	53	1232	3436	6355	53	1235	3436	6355	3	118	576	957	1	2	5	8
hep-th	8,361	15,751	19	50	2,500	23	50	179	527	23	50	186	544	1	1	2	10	1	1	1	1
whitaker3	9,800	28,989	161	8	64	4	10	19	31	4	10	19	31	1	1	2	2	1	1	1	1
crack	10,240	30,380	107	9	81	4	12	25	42	4	12	25	42	1	1	2	3	1	1	1	1
PGPgiantcompo	10,680	24,316	24	205	42,025	31	205	423	1160	31	205	424	1160	1	2	9	47	1	1	1	1
p2p-Gnutella04	10,876	39,994	10	103	10,609	7	103	686	3795	7	103	861	3979	1	3	71	750	1	1	1	4
astro-ph	16,706	$1.2{\cdot}10^5$	14	360	$1.3{\cdot}10^5$	56	360	2047	5846	56	364	2082	5870	2	29	515	2302	1	1	4	11
cond-mat	16,726	47,594	18	107	11,449	17	107	342	1151	17	107	351	1187	1	2	14	106	1	1	1	2
memplus	17,758	54,196	12	573	$3.3{\cdot}10^5$	96	573	8056	8962	96	573	8056	8962	2	32	648	2864	1	1	7	21
cs4	22,499	43,858	75	4	16	3	9	19	37	3	9	19	38	2	2	3	4	1	1	1	1
p2p-Gnutella25	22,687	54,705	11	66	4,356	5	66	327	2163	5	66	421	2618	2	4	38	1116	1	1	1	6
as-22july06	22,963	48,436	11	2,390	$5.7{\cdot}10^6$	25	2390	8920	15786	25	2390	8921	15788	3	132	3053	9617	1	2	21	101
p2p-Gnutella24	26,518	65,369	11	355	$1.3{\cdot}10^5$	5	355	422	4281	5	355	550	4281	2	6	92	2263	1	1	2	10
cit-HepTh	27,770	$3.5{\cdot}10^5$	15	2,468	$6.1{\cdot}10^6$	37	2468	8120	$\geq$14737	37	2468	8155	14823	7	866	11669	-	1	9	40	138
cond-mat-2003	31,163	$1.2{\cdot}10^5$	16	202	40,804	24	202	1215	4256	24	202	1252	4338	3	14	283	2874	1	1	3	11
cit-HepPh	34,546	$4.2{\cdot}10^5$	14	846	$7.2{\cdot}10^5$	30	846	4537	$\geq$13847	30	846	4651	15521	6	403	13427	-	1	6	40	129
p2p-Gnutella30	36,682	88,328	11	55	3,025	7	55	356	2609	7	55	473	3273	3	7	96	3130	1	1	2	11
cond-mat-2005	40,421	$1.8{\cdot}10^5$	18	278	77,284	29	278	1995	7101	29	278	2033	7214	4	39	1093	10813	1	2	7	28

Table A.4: Results for computing the weak/strong x-degeneracies of real-world graphs (part 4). We use the following notation: n–number of vertices, m–number of edges, dia–diameter, Δ–maximum degree, $x \in [1,4]$, α_x–weak x-degeneracy, β_x–strong x-degeneracy, t_{α_x}–time (in seconds) to compute weak x-degeneracy (set to "-" if timeout after 6 hours), t_{β_x}–time (in seconds) to compute strong x-degeneracy (set to "-" if timeout after 24 hours). See Appendix A on page 103 for more details.

Graph	n	m	dia	Δ	Δ^2	α_1	α_2	α_3	α_4	β_1	β_2	β_3	β_4	t_{α_1}	t_{α_2}	t_{α_3}	t_{α_4}	t_{β_1}	t_{β_2}	t_{β_3}	t_{β_4}
p2p-Gnutella31	62,586	$1.5{\cdot}10^5$	11	95	9,025	6	95	406	3226	6	95	532	4142	8	16	221	8915	2	2	4	25
kron500-16	65,536	$2.5{\cdot}10^6$	6	17,997	$3.2{\cdot}10^8$	432	$\geq$4306	$\geq$32208	$\geq$54989	432	17997	48805	54989	1616	-	-	-	15	710	2889	3543
soc-Epinions1	75,879	$4.1{\cdot}10^5$	15	3,044	$9.3{\cdot}10^6$	67	3044	$\geq$6251	$\geq$13628	67	3044	14719	41918	32	4894	-	-	3	25	188	1664
soc-Slashdot0811	77,360	$4.7{\cdot}10^5$	12	2,539	$6.4{\cdot}10^6$	54	2539	$\geq$6042	$\geq$17860	54	2539	17774	50926	22	7102	-	-	3	29	268	2779
graph_thres_05	81,519	$1.1{\cdot}10^5$	40	71	5,041	10	71	149	440	10	71	149	440	11	13	21	57	3	3	3	3
soc-Slashdot0902	82,168	$5 \cdot 10^5$	13	2,552	$6.5{\cdot}10^6$	55	2552	$\geq$5471	$\geq$16481	55	2552	18119	52568	24	7790	-	-	4	31	296	3051
prefAtt	$1 \cdot 10^5$	$5 \cdot 10^5$	7	983	$9.7{\cdot}10^5$	5	983	$\geq$3375	$\geq$22964	5	983	6619	43737	21	368	-	-	4	7	110	1181
smallworld	$1 \cdot 10^5$	$5 \cdot 10^5$	10	17	289	7	34	190	1006	7	36	214	1156	21	35	137	1121	5	5	8	20
G_n_pin_pout	$1 \cdot 10^5$	$5 \cdot 10^5$	9	25	625	7	64	587	$\geq$5248	7	71	698	6358	20	53	594	-	5	6	15	113
graph_thres_04	$1.1{\cdot}10^5$	$1.7{\cdot}10^5$	31	88	7,744	13	88	172	617	13	88	172	659	20	25	55	276	5	5	5	7
luxembourg.osm	$1.1{\cdot}10^5$	$1.2{\cdot}10^5$	1,337	6	36	2	6	10	16	2	6	10	16	23	22	26	25	6	6	5	5
rgg_n_2_17_s0	$1.3{\cdot}10^5$	$7.3{\cdot}10^5$	341	28	784	14	29	47	71	14	29	47	71	32	44	63	92	7	7	8	8
wave	$1.6{\cdot}10^5$	$1.1{\cdot}10^6$	56	44	1,936	8	44	114	255	8	44	114	258	45	97	237	550	10	11	13	15
graph_thres_03	$1.7{\cdot}10^5$	$2.9{\cdot}10^5$	29	123	15,129	16	123	313	1701	16	123	334	1802	44	62	243	2784	10	10	12	21
caidaRouterLevel	$1.9{\cdot}10^5$	$6.1{\cdot}10^5$	26	1,071	$1.1{\cdot}10^6$	32	1071	3855	$\geq$1539	32	1071	3921	15812	64	427	16926	-	13	16	53	308
coAuthorsCiteseer	$2.3{\cdot}10^5$	$8.1{\cdot}10^5$	33	1,372	$1.9{\cdot}10^6$	86	1372	1633	$\geq$2922	86	1372	1633	6818	86	209	2029	-	18	19	28	99
amazon0302	$2.6{\cdot}10^5$	$9 \cdot 10^5$	38	420	$1.8{\cdot}10^5$	6	420	838	3611	6	420	839	3611	116	205	1157	14694	27	26	33	76
email-EuAll	$2.7{\cdot}10^5$	$3.6{\cdot}10^5$	14	7,636	$5.8{\cdot}10^7$	37	7636	$\geq$2795	$\geq$13373	37	7636	19371	$\geq$162540	129	8781	-	-	28	50	566	-
citationCiteseer	$2.7{\cdot}10^5$	$1.2{\cdot}10^6$	36	1,318	$1.7{\cdot}10^6$	15	1318	$\geq$2215	$\geq$2017	15	1318	3350	18245	123	1048	-	-	25	34	152	1022
web-Stanford	$2.8{\cdot}10^5$	$2 \cdot 10^6$	753	38,625	$1.5{\cdot}10^9$	71	$\geq$19004	$\geq$3672	$\geq$4086	71	38625	42983	76672	362	-	-	-	33	1379	3254	9473
graph_thres_02	$2.8{\cdot}10^5$	$6.4{\cdot}10^5$	29	201	40,401	63	201	1036	$\geq$2041	63	201	1097	7073	124	239	3029	-	28	28	41	151
coAuthorsDBLP	$3 \cdot 10^5$	$9.8{\cdot}10^5$	24	336	$1.1{\cdot}10^5$	114	336	1982	$\geq$1398	114	336	1982	12057	146	435	12319	-	30	32	70	426
cnr-2000	$3.3{\cdot}10^5$	$2.7{\cdot}10^6$	34	18,236	$3.3{\cdot}10^8$	83	$\geq$17814	$\geq$6425	$\geq$6372	83	18236	24048	44759	301	-	-	-	45	211	480	2794
web-NotreDame	$3.3{\cdot}10^5$	$1.1{\cdot}10^6$	46	10,721	$1.1{\cdot}10^8$	155	10721	$\geq$3809	$\geq$3532	155	10721	12019	56853	177	8435	-	-	38	66	194	3002
amazon0312	$4 \cdot 10^5$	$2.3{\cdot}10^6$	20	2,747	$7.5{\cdot}10^6$	10	2747	$\geq$1418	$\geq$1356	10	2747	5343	22844	268	1915	-	-	60	69	219	1698
amazon0601	$4 \cdot 10^5$	$2.4{\cdot}10^6$	25	2,752	$7.6{\cdot}10^6$	10	2752	$\geq$1374	$\geq$1437	10	2752	5838	22131	273	2007	-	-	62	71	234	1761
amazon0505	$4.1{\cdot}10^5$	$2.4{\cdot}10^6$	22	2,760	$7.6{\cdot}10^6$	10	2760	$\geq$1328	$\geq$1341	10	2760	5864	24349	281	2090	-	-	63	73	241	1863
coPapersCiteseer	$4.3{\cdot}10^5$	$1.6{\cdot}10^7$	34	1,188	$1.4{\cdot}10^6$	844	1220	$\geq$976	$\geq$826	844	1238	7666	32700	803	13345	-	-	69	128	693	3763
rgg_n_2_19_s0	$5.2{\cdot}10^5$	$3.3{\cdot}10^6$	633	30	900	17	31	53	80	17	31	53	80	450	665	1002	1498	101	105	108	110
coPapersDBLP	$5.4{\cdot}10^5$	$1.5{\cdot}10^7$	23	3,299	$1.1{\cdot}10^7$	336	$\geq$1184	$\geq$805	$\geq$687	336	3299	14736	73759	942	-	-	-	101	222	1880	12259
web-BerkStan	$6.9{\cdot}10^5$	$6.6{\cdot}10^6$	714	84,230	$7.1{\cdot}10^9$	201	$\geq$5665	$\geq$1827	$\geq$940	201	84230	93371	$\geq$140580	1787	-	-	-	190	10557	25191	-
graph_thres_01	$7.2{\cdot}10^5$	$2.5{\cdot}10^6$	22	804	$6.5{\cdot}10^5$	113	804	$\geq$531	$\geq$517	113	804	8411	52089	789	4772	-	-	174	187	669	6667
eu-2005	$8.6{\cdot}10^5$	$1.6{\cdot}10^7$	21	68,963	$4.8{\cdot}10^9$	388	$\geq$2140	$\geq$2009	$\geq$1470	388	68963	116230	$\geq$377849	2986	-	-	-	291	11182	64403	-
web-Google	$8.8{\cdot}10^5$	$4.3{\cdot}10^6$	24	6,332	$4 \cdot 10^7$	44	$\geq$3120	$\geq$1319	$\geq$311	44	6332	7696	$\geq$259559	1290	-	-	-	261	362	801	-
ldoor	$9.5{\cdot}10^5$	$2.3{\cdot}10^7$	186	76	5,776	34	90	188	321	34	90	188	321	2520	5677	10892	18739	349	356	367	381
ecology1	$1 \cdot 10^6$	$2 \cdot 10^6$	1,998	4	16	2	6	12	20	2	6	12	20	1490	1761	2223	2785	450	410	427	444
rgg_n_2_20_s0	$1 \cdot 10^6$	$6.9{\cdot}10^6$	865	36	1,296	17	37	59	88	17	37	59	88	1776	2727	4194	6288	405	419	429	434
in-2004	$1.4{\cdot}10^6$	$1.4{\cdot}10^7$	43	21,869	$4.8{\cdot}10^8$	488	$\geq$2907	$\geq$973	$\geq$505	488	21869	24312	81579	3891	-	-	-	644	1322	1981	16680
belgium.osm	$1.4{\cdot}10^6$	$1.5{\cdot}10^6$	1,987	10	100	3	10	14	21	3	10	14	21	3208	3189	3514	3347	837	793	763	747

Table A.5: Results for computing the gaps for Maximum Clique relaxations on real-world graphs (part 1). The graphs are sorted ascending by their number of vertices. We use the following notation: $s \in [1,4]$, α_x–weak x-degeneracy, CsDC–connected s-defective clique, sDC–s-defective clique, CsP–connected s-plex, sP–s-plex, sC–s-club, ω_Π–maximum-order of a subgraph satisfying Π (taken from the literature), t_Π–time to compute ω_Π (in seconds, taken from the literature), $g_{\mathrm{CsDC}} := \alpha_{\left\lfloor \sqrt{2s+\frac{1}{4}}+\frac{1}{2} \right\rfloor} + 1 - \omega_{\mathrm{sDC}}$, $g_{\mathrm{CsP}} := \alpha_s + 1 - \omega_{\mathrm{sP}}$, $g_{\mathrm{sC}} := \alpha_s + 1 - \omega_{\mathrm{sC}}$, cite–sources for the respective s-club-values. If a maximum-order s-plex is not guaranteed to be connected, then we set g_{CsP} to "?". See Appendix A on page 103 for more details.

Graph	ω_{1DC}	g_{C1DC}	t_{1DC}	ω_{2DC}	g_{C2DC}	t_{2DC}	ω_{3DC}	g_{C3DC}	t_{3DC}	ω_{4DC}	g_{C4DC}	t_{4DC}	ω_{2P}	g_{C2P}	t_{2P}	ω_{3P}	g_{C3P}	t_{3P}	ω_{4P}	g_{C4P}	t_{4P}	ω_{2C}	g_{2C}	t_{2C}	ω_{3C}	g_{3C}	t_{3C}	ω_{4C}	g_{4C}	t_{4C}	cite
johnson8-2-4	4	24	0	5	23	0	5	23	0	6	22	0	5	23	0	8	20	0	9	19	0	-	-	-	-	-	-	-	-	-	-
karate	6	12	0	6	12	0	6	19	1	6	19	1	6	12	0	6	19	0	8	25	0	18	0	1	-	-	-	-	-	-	S
chesapeake	6	30	0	6	30	1	7	32	1	7	32	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MANN_a9	17	28	1	18	27	1	19	26	1	20	25	2	26	19	0	36	9	0	36	9	0	-	-	-	-	-	-	-	-	-	-
dolphins	6	11	0	6	11	0	6	23	1	7	22	1	6	11	0	7	22	0	7	33	0	13	4	1	29	0	1	40	0	1	S
hamming6-2	32	32	0	32	32	0	32	32	0	32	32	0	32	32	0	32	32	0	40	24	0	-	-	-	-	-	-	-	-	-	-
hamming6-4	4	60	0	5	59	1	6	58	1	6	58	1	6	58	0	8	56	0	10	54	0	-	-	-	-	-	-	-	-	-	-
johnson8-4-4	14	56	0	14	56	1	14	56	1	15	55	1	14	56	0	18	52	0	22	48	0	-	-	-	-	-	-	-	-	-	-
lesmis	10	27	0	11	26	0	11	47	0	12	46	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
polbooks	7	22	1	7	22	1	8	45	1	8	45	1	7	22	0	9	44	0	10	60	0	28	1	1	53	0	1	68	2	1	S
adjnoun	6	44	1	6	44	1	7	81	1	7	81	2	6	44	0	8	80	0	8	99	0	50	0	0	82	6	1	-	-	-	K,S
football	9	29	1	9	29	1	9	89	1	9	89	1	10	28	1	11	87	1	12	103	0	16	22	1	58	40	1	-	-	-	S
johnson16-2-4	8	112	21	9	111	244	9	111	289	10	110	345	10	110	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
C125.9	35	90	38	36	89	379	37	88	1887	38	87	4663	43	82	0	51	74	2	-	-	-	125	0	1	-	-	-	-	-	-	H
keller4	12	159	3	13	158	47	14	157	439	15	156	637	15	156	0	21	150	1	-	-	-	171	0	1	-	-	-	-	-	-	H
jazz	30	80	0	30	80	0	30	145	0	30	145	0	30	80	0	30	145	0	30	162	0	103	7	1	174	1	1	-	-	-	S
brock200_1	21	179	540	22	178	6911	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
brock200_2	12	188	2	12	188	24	13	187	109	13	187	831	13	187	1	16	184	1	-	-	-	200	0	1	-	-	-	-	-	-	H
brock200_3	15	185	27	16	184	386	16	184	2729	17	183	13749	17	183	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
brock200_4	17	183	56	18	182	804	18	182	6604	-	-	-	20	180	1	-	-	-	-	-	-	200	0	1	-	-	-	-	-	-	H
c-fat200-1	12	13	0	12	13	1	12	23	1	12	23	1	12	13	0	12	23	0	12	33	0	-	-	-	-	-	-	-	-	-	-
c-fat200-2	24	31	0	24	31	0	24	53	1	24	53	1	24	31	1	24	53	1	24	75	1	-	-	-	-	-	-	-	-	-	-
c-fat200-5	58	84	0	58	84	0	58	142	1	58	142	1	58	84	1	58	142	1	58	142	0	-	-	-	-	-	-	-	-	-	-
san200_0.7_1	30	170	5	30	170	71	30	170	729	30	170	5227	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
san200_0.7_2	19	181	5	19	181	99	20	180	1002	20	180	8621	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
san200_0.9_1	70	130	12	70	130	73	71	129	1296	71	129	2454	90	110	1	125	75	1	125	75	1	-	-	-	-	-	-	-	-	-	-
san200_0.9_2	60	140	96	61	139	1345	61	139	2296	61	139	1693	-	-	-	105	95	1	105	95	1	-	-	-	-	-	-	-	-	-	-
san200_0.9_3	44	156	192	45	155	380	45	155	4827	46	154	2892	-	-	-	-	-	-	96	104	1	-	-	-	-	-	-	-	-	-	-
sanr200_0.7	19	181	152	19	181	2329	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
sanr200_0.9	43	157	5046	44	156	6683	45	155	12502	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
gen200_p0.9_44	45	155	697	46	154	153	46	154	279	47	153	4791	-	-	-	-	-	-	-	-	-	200	0	1	-	-	-	-	-	-	H
gen200_p0.9_55	56	144	158	57	143	2527	57	143	167	58	142	174	-	-	-	-	-	-	-	-	-	200	0	1	-	-	-	-	-	-	H
C250.9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	250	0	1	-	-	-	-	-	-	H
hamming8-2	128	128	1	128	128	1	128	128	1	128	128	1	128	128	1	128	128	1	-	-	-	-	-	-	-	-	-	-	-	-	-
hamming8-4	16	240	1	16	240	3	16	240	64	17	239	529	16	240	1	-	-	-	-	-	-	256	0	1	-	-	-	-	-	-	H
celegansneural	8	144	1	9	143	2	10	255	39	10	255	40	-	-	-	-	-	-	-	-	-	285	-133	1	243	22	1	-	-	-	H,S
p_hat300-1	9	290	1	9	290	4	10	290	21	10	290	136	10	289	0	12	288	0	14	286	1	299	0	1	-	-	-	-	-	-	H
p_hat300-2	26	274	18	27	273	324	28	272	3765	-	-	-	30	270	1	-	-	-	-	-	-	300	0	1	-	-	-	-	-	-	H

Table A.6: Results for computing the gaps for Maximum Clique relaxations on real-world graphs (part 2). The graphs are sorted ascending by their number of vertices. We use the following notation: $s \in [1, 4]$, α_x–weak x-degeneracy, CsDC–connected s-defective clique, sDC–s-defective clique, CsP–connected s-plex, sP–s-plex, sC–s-club, ω_Π–maximum-order of a subgraph satisfying Π (taken from the literature), t_Π–time to compute ω_Π (in seconds, taken from the literature), $g_{\text{CsDC}} := \alpha_{\left\lfloor\sqrt{2s+\frac{1}{4}}+\frac{1}{2}\right\rfloor} + 1 - \omega_{\text{sDC}}$, $g_{\text{CsP}} := \alpha_s + 1 - \omega_{\text{sP}}$, $g_{\text{sC}} := \alpha_s + 1 - \omega_{\text{sC}}$, cite–sources for the respective s-club-values. If a maximum-order s-plex is not guaranteed to be connected, then we set g_{CsP} to "?". See Appendix A on page 103 for more details.

Graph	$\omega_{1\text{DC}}$	g_{C1DC}	$t_{1\text{DC}}$	$\omega_{2\text{DC}}$	g_{C2DC}	$t_{2\text{DC}}$	$\omega_{3\text{DC}}$	g_{C3DC}	$t_{3\text{DC}}$	$\omega_{4\text{DC}}$	g_{C4DC}	$t_{4\text{DC}}$	$\omega_{2\text{P}}$	g_{C2P}	$t_{2\text{P}}$	$\omega_{3\text{P}}$	g_{C3P}	$t_{3\text{P}}$	$\omega_{4\text{P}}$	g_{C4P}	$t_{4\text{P}}$	$\omega_{2\text{C}}$	$g_{2\text{C}}$	$t_{2\text{C}}$	$\omega_{3\text{C}}$	$g_{3\text{C}}$	$t_{3\text{C}}$	$\omega_{4\text{C}}$	$g_{4\text{C}}$	$t_{4\text{C}}$	cite
p_hat300-3	37	263	1799	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	300	0	1	-	-	-	-	-	-	H
MANN_a27	127	251	1163	128	250	111	129	249	126	130	248	134	236	142	13	351	27	1	-	-	-	378	0	1	-	-	-	-	-	-	H
brock400_1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
brock400_2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	400	0	1	-	-	-	-	-	-	H
brock400_3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
brock400_4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	400	0	1	-	-	-	-	-	-	H
san400_0.5_1	13	387	29	13	387	1115	13	387	13966	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
san400_0.7_1	40	360	396	40	360	11190	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
san400_0.7_2	30	370	554	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
san400_0.7_3	22	378	2890	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
san400_0.9_1	100	300	3182	100	300	2626	100	300	5322	100	300	3645	-	-	-	-	-	-	200	200	1	-	-	-	-	-	-	-	-	-	-
sanr400_0.5	14	386	145	14	386	3395	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
sanr400_0.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
gen400_p0.9_55	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	400	0	1	-	-	-	-	-	-	H
gen400_p0.9_65	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	400	0	1	-	-	-	-	-	-	H
gen400_p0.9_75	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	400	0	1	-	-	-	-	-	-	H
celegans_metabolic	10	228	1	10	228	1	11	362	7	11	362	56	10	228	0	11	362	0	13	419	0	238	0	1	371	2	1	432	0	1	K,S
johnson32-2-4	16	480	1814	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
c-fat500-1	14	16	1	14	16	1	14	28	1	14	28	2	14	16	0	14	28	0	14	40	0	-	-	-	-	-	-	-	-	-	-
c-fat500-2	26	34	1	26	34	1	26	58	1	26	58	1	26	34	0	26	58	0	26	82	0	-	-	-	-	-	-	-	-	-	-
c-fat500-5	64	91	1	64	91	1	64	153	1	64	153	1	64	91	1	64	153	1	64	215	1	-	-	-	-	-	-	-	-	-	-
c-fat500-10	126	185	1	126	185	1	126	311	1	126	311	1	126	185	1	126	311	1	126	374	1	-	-	-	-	-	-	-	-	-	-
p_hat500-1	10	490	4	11	489	46	11	489	495	12	488	2738	12	488	1	14	486	1	-	-	-	-	-	-	-	-	-	-	-	-	-
p_hat500-2	37	463	457	38	462	10172	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
p_hat500-3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DSJC500.5	14	486	1029	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	500	0	1	-	-	-	-	-	-	H
C500.9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	500	0	1	-	-	-	-	-	-	H
p_hat700-1	12	688	10	12	688	189	13	687	1321	13	687	12478	13	687	1	-	-	-	-	-	-	700	0	1	-	-	-	-	-	-	H
p_hat700-2	45	655	4724	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	700	0	2	-	-	-	-	-	-	H
p_hat700-3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	700	0	2	-	-	-	-	-	-	H
keller5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	776	0	3	-	-	-	-	-	-	H
brock800_1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
brock800_2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	800	0	3	-	-	-	-	-	-	H
brock800_3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
brock800_4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	800	0	3	-	-	-	-	-	-	H
p_hat1000-1	11	989	128	12	988	2008	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table A.7: Results for computing the gaps for MAXIMUM CLIQUE relaxations on real-world graphs (part 3). The graphs are sorted ascending by their number of vertices. We use the following notation: $s \in [1, 4]$, α_x–weak x-degeneracy, CsDC–connected s-defective clique, sDC–s-defective clique, CsP–connected s-plex, sP–s-plex, sC–s-club, ω_Π–maximum-order of a subgraph satisfying Π (taken from the literature), t_Π–time to compute ω_Π (in seconds, taken from the literature), $g_{\text{CsDC}} := \alpha_{\left\lfloor \sqrt{2s+\frac{1}{4}}+\frac{1}{2} \right\rfloor} + 1 - \omega_{\text{sDC}}$, $g_{\text{CsP}} := \alpha_s + 1 - \omega_{\text{sP}}$, $g_{\text{sC}} := \alpha_s + 1 - \omega_{\text{sC}}$, cite–sources for the respective s-club-values. If a maximum-order s-plex is not guaranteed to be connected, then we set g_{CsP} to "?". See Appendix A on page 103 for more details.

Graph	$\omega_{1\text{DC}}$	g_{C1DC}	$t_{1\text{DC}}$	$\omega_{2\text{DC}}$	g_{C2DC}	$t_{2\text{DC}}$	$\omega_{3\text{DC}}$	g_{C3DC}	$t_{3\text{DC}}$	$\omega_{4\text{DC}}$	g_{C4DC}	$t_{4\text{DC}}$	$\omega_{2\text{P}}$	g_{C2P}	$t_{2\text{P}}$	$\omega_{3\text{P}}$	g_{C3P}	$t_{3\text{P}}$	$\omega_{4\text{P}}$	g_{C4P}	$t_{4\text{P}}$	$\omega_{2\text{C}}$	$g_{2\text{C}}$	$t_{2\text{C}}$	$\omega_{3\text{C}}$	$g_{3\text{C}}$	$t_{3\text{C}}$	$\omega_{4\text{C}}$	$g_{4\text{C}}$	$t_{4\text{C}}$	cite
p_hat1000-2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
p_hat1000-3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
san1000	15	985	1901	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DSJC1000.5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1000	0	3	-	-	-	-	-	-	H
hamming10-2	512	512	2	512	512	3	512	512	3	512	512	4	512	512	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
hamming10-4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MANN_a45	346	689	641	347	688	2247	348	687	2289	349	686	1074	662	373	6	990	45	8	990	45	8	-	-	-	-	-	-	-	-	-	-
email	12	76	0	12	76	1	12	373	1	13	372	7	12	76	1	12	373	1	12	790	1	72	16	4	212	173	122	651	151	2	S
polblogs	21	356	7	22	355	16	22	843	532	23	842	3007	23	354	1	27	838	1	29	1111	1	352	25	5	776	89	2	1127	13	1	S
p_hat1500-1	12	1488	2035	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1500	0	27	-	-	-	-	-	-	H
p_hat1500-2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
p_hat1500-3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
netscience	20	15	0	20	15	0	20	34	0	20	34	0	20	15	0	20	34	0	20	65	0	35	0	1	-	-	-	-	-	-	K
add20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	124	0	1	-	-	-	-	-	-	K
data	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	18	6	4	32	14	6	52	22	9	S
MANN_a81	-	-	-	-	-	-	-	-	-	-	-	-	2162	1159	140	3240	81	139	3240	81	148	-	-	-	-	-	-	-	-	-	-
keller6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
uk	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	2	4	8	4	5	14	4	5	S
power	6	14	0	6	14	0	7	23	1	-	-	-	6	14	0	6	24	0	8	53	1	20	0	1	-	-	-	-	-	-	K
add32	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	32	0	1	-	-	-	-	-	-	S
wiki-Vote	-	-	-	-	-	-	-	-	-	-	-	-	21	1212	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
hep-th	24	27	0	24	27	0	24	156	0	24	156	0	-	-	-	-	-	-	-	-	-	51	0	1	120	60	48	344	184	405	K,S
whitaker3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9	2	21	15	5	25	23	9	30	S
crack	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10	3	26	17	9	29	31	12	38	S
PGPgiantcompo	26	180	1	27	179	1	28	396	1	28	396	5	29	177	1	31	393	1	33	1128	1	206	0	1	422	2	6	-	-	-	K,S
p2p-Gnutella04	4	100	15231	-	-	-	-	-	-	-	-	-	5	99	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
astro-ph	57	304	0	57	304	0	57	1991	1	57	1991	1	57	304	1	57	1991	1	57	5790	1	-	-	-	-	-	-	-	-	-	-
cond-mat	18	90	0	18	90	0	18	325	0	18	325	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
memplus	97	477	0	97	477	0	97	7960	0	97	7960	0	97	477	1	97	7960	1	97	8866	1	-	-	-	-	-	-	-	-	-	-
cs4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	4	84	12	8	90	18	20	114	S
p2p-Gnutella25	-	-	-	-	-	-	-	-	-	-	-	-	5	62	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
as-22july06	18	2373	1	18	2373	2	19	8902	32	19	8902	76	19	2372	1	21	8900	1	22	15765	1	-	-	-	-	-	-	-	-	-	-
p2p-Gnutella24	-	-	-	-	-	-	-	-	-	-	-	-	5	351	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
cit-HepTh	-	-	-	-	-	-	-	-	-	-	-	-	28	2441	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
cond-mat-2003	25	178	0	25	178	0	26	1190	0	26	1190	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
cit-HepPh	-	-	-	-	-	-	-	-	-	-	-	-	24	823	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
p2p-Gnutella30	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
cond-mat-2005	30	249	0	30	249	0	30	1966	0	30	1966	0	30	249	1	30	1966	1	30	7072	1	-	-	-	-	-	-	-	-	-	-

Table A.8: Results for computing the gaps for MAXIMUM CLIQUE relaxations on real-world graphs (part 4). The graphs are sorted ascending by their number of vertices. We use the following notation: $s \in [1,4]$, α_x–weak x-degeneracy, CsDC–connected s-defective clique, sDC–s-defective clique, CsP–connected s-plex, sP–s-plex, sC–s-club, ω_Π–maximum-order of a subgraph satisfying Π (taken from the literature), t_Π–time to compute ω_Π (in seconds, taken from the literature), $g_{\mathrm{CsDC}} := \alpha_{\left\lfloor\sqrt{2s+\frac{1}{4}}+\frac{1}{2}\right\rfloor} + 1 - \omega_{\mathrm{sDC}}$, $g_{\mathrm{CsP}} := \alpha_s + 1 - \omega_{\mathrm{sP}}$, $g_{\mathrm{sC}} := \alpha_s + 1 - \omega_{\mathrm{sC}}$, cite–sources for the respective s-club-values. If a maximum-order s-plex is not guaranteed to be connected, then we set g_{CsP} to "?". See Appendix A on page 103 for more details.

Graph	ω_{1DC}	g_{C1DC}	t_{1DC}	ω_{2DC}	g_{C2DC}	t_{2DC}	ω_{3DC}	g_{C3DC}	t_{3DC}	ω_{4DC}	g_{C4DC}	t_{4DC}	ω_{2P}	g_{C2P}	t_{2P}	ω_{3P}	g_{C3P}	t_{3P}	ω_{4P}	g_{C4P}	t_{4P}	ω_{2C}	g_{2C}	t_{2C}	ω_{3C}	g_{3C}	t_{3C}	ω_{4C}	g_{4C}	t_{4C}	cite
p2p-Gnutella31	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
kron500-16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
soc-Epinions1	24	3021	16030	-	-	-	-	-	-	-	-	-	28	3017	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
soc-Slashdot0811	27	2513	15695	-	-	-	-	-	-	-	-	-	31	2509	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
graph_thres_05	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	72	0	9	-	-	-	-	-	-	H
soc-Slashdot0902	28	2525	15944	-	-	-	-	-	-	-	-	-	32	2521	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
prefAtt	-	-	-	-	-	-	-	-	-	-	-	-	7	977	1	8	≥3368	1	9	≥22956	1	-	-	-	-	-	-	-	-	-	-
smallworld	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
G_n_pin_pout	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
graph_thres_04	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	89	0	14	-	-	-	-	-	-	H
luxembourg.osm	-	-	-	-	-	-	-	-	-	-	-	-	4	3	1	5	6	1	6	?	1	-	-	-	-	-	-	-	-	-	-
rgg_n_2_17_s0	15	15	0	16	14	1	16	32	2	-	-	-	16	14	1	17	31	1	18	54	1	-	-	-	-	-	-	-	-	-	-
wave	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
graph_thres_03	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	124	0	25	-	-	-	-	-	-	H
caidaRouterLevel	18	1054	3314	-	-	-	-	-	-	-	-	-	20	1052	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
coAuthorsCiteseer	87	1286	0	87	1286	0	87	1547	0	87	1547	0	87	1286	1	87	1547	1	87	≥2836	1	1373	0	12	-	-	-	-	-	-	K
amazon0302	-	-	-	-	-	-	-	-	-	-	-	-	8	413	1	9	830	1	10	3602	1	-	-	-	-	-	-	-	-	-	-
email-EuAll	17	7620	443	17	7620	12317	-	-	-	-	-	-	19	7618	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
citationCiteseer	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1319	0	22	-	-	-	-	-	-	K
web-Stanford	62	≥18943	49	63	≥18942	212	64	≥3609	8284	-	-	-	64	≥18941	4	64	≥3609	6	-	-	-	-	-	-	-	-	-	-	-	-	-
graph_thres_02	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	202	0	60	-	-	-	-	-	-	H
coAuthorsDBLP	115	222	0	115	222	0	115	1868	0	115	1868	0	115	222	1	115	1868	1	115	≥1284	1	337	0	16	-	-	-	-	-	-	K
cnr-2000	85	≥17730	0	85	≥17730	0	86	≥6340	0	86	≥6340	1	85	≥17730	7	86	≥6340	11	86	≥6287	8	-	-	-	-	-	-	-	-	-	-
web-NotreDame	155	10567	519	155	10567	1360	-	-	-	-	-	-	155	10567	1	155	≥3655	1	155	≥3378	1	-	-	-	-	-	-	-	-	-	-
amazon0312	-	-	-	-	-	-	-	-	-	-	-	-	12	2736	1	13	≥1406	1	14	≥1343	1	-	-	-	-	-	-	-	-	-	-
amazon0601	-	-	-	-	-	-	-	-	-	-	-	-	12	2741	1	13	≥1362	1	14	≥1424	1	-	-	-	-	-	-	-	-	-	-
amazon0505	-	-	-	-	-	-	-	-	-	-	-	-	12	2749	1	13	≥1316	1	14	≥1328	1	-	-	-	-	-	-	-	-	-	-
coPapersCiteseer	-	-	-	-	-	-	-	-	-	-	-	-	845	376	8	845	≥132	8	845	≥-18	8	-	-	-	-	-	-	-	-	-	-
rgg_n_2_19_s0	19	13	0	19	13	0	19	35	1	20	34	1	19	13	1	19	35	1	20	61	1	-	-	-	-	-	-	-	-	-	-
coPapersDBLP	-	-	-	-	-	-	-	-	-	-	-	-	337	≥848	3	337	≥469	4	337	≥351	4	3300	≥-2115	251	-	-	-	-	-	-	K
web-BerkStan	202	≥5464	1	202	≥5464	2	202	≥1626	7	202	≥1626	33	202	≥5464	5	202	≥1626	5	202	≥739	5	-	-	-	-	-	-	-	-	-	-
graph_thres_01	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	805	0	76	-	-	-	-	-	-	K
eu-2005	-	-	-	-	-	-	-	-	-	-	-	-	388	≥1753	5	390	≥1620	6	391	≥1080	8	-	-	-	-	-	-	-	-	-	-
web-Google	45	≥3076	0	46	≥3075	1	46	≥1274	1	47	≥1273	1	46	≥3075	2	47	≥1273	2	48	≥264	2	-	-	-	-	-	-	-	-	-	-
ldoor	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ecology1	-	-	-	-	-	-	-	-	-	-	-	-	4	3	0	-	-	-	6	?	1	-	-	-	-	-	-	-	-	-	-
rgg_n_2_20_s0	18	20	0	18	20	1	-	-	-	-	-	-	18	20	2	19	41	2	20	69	2	-	-	-	-	-	-	-	-	-	-
in-2004	-	-	-	-	-	-	-	-	-	-	-	-	490	≥2418	4	491	≥483	7	491	≥15	8	-	-	-	-	-	-	-	-	-	-
belgium.osm	-	-	-	-	-	-	-	-	-	-	-	-	5	6	1	5	10	1	6	?	1	-	-	-	-	-	-	-	-	-	-