On Hard Subgraph Problems: Parameterized Algorithms and Efficient Implementations

DISSERTATION
FOR THE DEGREE OF DOCTOR OF NATURAL SCIENCES
(DR. RER. NAT.)

Submitted by
Frank Sommer, M.Sc.
born October 29, 1993 in Jena, Germany.

Referees:
Prof. Dr. Christian Komusiewicz, Philipps-Universität Marburg, Germany
Prof. Dr. Holger Dell, Goethe-Universität Frankfurt, Germany

Examination Committee:
Prof. Dr. Holger Dell
Prof. Dr. Dominik Heider (Chairman)
Prof. Dr. Christian Komusiewicz
Prof. Dr. Thorsten Papenbrock
Prof. Dr. Elmar Tischhauser

Submitted on July 14, 2022
Defended on November 17, 2022
Philipps-Universität Marburg (1180)
Fachbereich 12 – Mathematik und Informatik
Marburg, Germany, 2022

Sommer, Frank
On Hard Subgraph Problems: Parameterized Algorithms and Efficient Implementations
Dissertation, Philipps-Universität Marburg, 2022.

Curriculum vitae

- Since November 2017: Member of the *Algorithmics* group
  Philipps-Universität Marburg, Germany
- October 2015 – October 2017: M.Sc. in *Mathematics*
  Friedrich-Schiller-Universität Jena, Germany.
- October 2012 – September 2015: B.Sc. in *Mathematics*
  Friedrich-Schiller-Universität Jena, Germany.

Preface

This thesis summarizes my results on (connected) subgraph problems. I study the complexity of these problems and I provide efficient algorithms for them. The results contained in this thesis were obtained from November 2017 to June 2022 at the Philipps-Universität Marburg at the Fachbereich Mathematik und Informatik in the Algorithmics research group lead by Christian Komusiewicz.

Most of the results presented in this thesis are contained in conference and journal publications that were developed in close collaboration with other coauthors. Before I roughly describe which chapters are based on which publications, I give a short overview on the other publications were I was a coauthor which are not included in this thesis. These publications are ordered alphabetically.

- "Approximation Algorithms for BalancedCC Multiwinner Rules", with Markus Brill, Piotr Faliszewski, and Nimrod Talmon. Conference: *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '19)* [33].
- "Colored Cut Games", with Nils Morawietz, Niels Grüttemeier, and Christian Komusiewicz. Journal: *Theoretical Computer Science* [175]. Conference: *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '20)* [173].
- "Destroying Bicolored P_3s by Deleting Few Edges", with Niels Grüttemeier, Christian Komusiewicz, and Jannik Schestag. Journal: *Discrete Mathematics & Theoretical Computer Science* [99]. Conference: *15th Conference on Computability in Europe (CIE '19)* [98].
- "Destroying Multicolored Paths and Cycles in Edge-Colored Graphs", with Nils Jakob Eckstein, Niels Grüttemeier, and Christian Komusiewicz. ArXiv: [65].
- "Essentially Tight Kernels For (Weakly) Closed Graphs" with Tomohiro Koana, and Christian Komusiewicz. Conference: *32nd International Symposium on Algorithms and Computation (ISAAC '21)* [140].
- "Exploiting c-Closure in Kernelization Algorithms for Graph Problems", with Tomohiro Koana, and Christian Komusiewicz. Journal: *SIAM Journal on Discrete Mathematics* [141]. Conference: *28th Annual European Symposium on Algorithms (ESA '20)* [139].
- "Multi-Parameter Analysis of Finding Minors and Subgraphs in Edge Periodic Temporal Graphs", with Emmanuel Arrighi, Niels Grüttemeier, Nils Morawietz, and Petra Wolf. Conference: Accepted at *48th International Conference on*

Current Trends in Theory and Practice of Computer Science (SOFSEM '23). ArXiv: [10].

- "On Critical Node Problems with Vulnerable Vertices", with Jannik Schestag, Niels Grüttemeier, and Christian Komusiewicz. Conference: *33rd International Workshop on Combinatorial Algorithms (IWOCA '22)* [207].
- "Preventing Small (s,t)-Cuts by Protecting Edges", with Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. Conference: *47th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '21)* [97].
- "Refined Parameterizations for Computing Colored Cuts in Edge-Colored Graphs", with Nils Morawietz, Niels Grüttemeier, and Christian Komusiewicz. Journal: *Theory of Computing Systems* [176]. Conference: *46th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '20)* [174].
- "String Factorizations Under Various Collision Constraints", with Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. Conference: *Thirty-First Annual Symposium on Combinatorial Pattern Matching (CPM '20)* [96].

In the following paragraphs, I describe which chapters are based on which publications. Furthermore, I describe my own contribution to these publications.

Chapter 3 is based on the publication "Enumerating connected induced subgraphs: Improved delay and experimental comparison" written with Christian Komusiewicz, which appeared in *Discrete Applied Mathematics* [148]. A preliminary version of this publication appeared in the *Proceedings of the 45th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '19)* [146]. Initially, the aim of this project initiated by Christian and myself was to find a suitable enumeration algorithm for `FixCon` (see Chapter 4). During this study I discovered that improving the current best delay for the problem of enumerating all connected induced subgraphs of size exactly k was possible by an adoption of one of the algorithms included in our study. Furthermore, after the publication of the conference version we came across with a similar problem: the enumeration of all connected induced subgraphs of size at most k [4]. We also improved the delay for this problem in the journal version. Both coauthors jointly worked out the details of the improved delay for enumerating all connected induced subgraphs of size exactly k. I worked out the details of the improved delay for the enumeration of all connected induced subgraphs of size at most k and I implemented all algorithms considered in this study and evaluated the results. Furthermore, I prepared the draft of the manuscript. Chapter 3 also contains some new results discovered by me regarding related enumeration problems which are not contained in the journal version.

More precisely, it contains also results for the delay of enumerating all edge-induced subgraphs, and the enumeration of all (not necessarily induced) subgraphs.

Chapter 4 is based on the publication "FixCon: A Generic Solver for Fixed-Cardinality Subgraph Problems" written with Christian Komusiewicz which appeared in the *Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX '20)* [147]. Christian proposed to implement `FixCon`, a generic solver for Connected Fixed-Cardinality Optimization. The pruning and data reduction to speed up the solver were jointly discovered by both coauthors. Christian implemented the neighborhood-based rules and the problem-specific pruning rules. I implemented the rest of the solver. Furthermore, Christian implemented the ILP formulations which we used to evaluate `FixCon`. I did the comparison of `FixCon` and the ILP formulations. The manuscript was prepared by both coauthors.

Chapter 5 is based on parts of the publication "Computing Dense and Sparse Subgraphs of Weakly Closed Graphs" written with Tomohiro Koana and Christian Komusiewicz which appeared in the *Proceedings of the 31st International Symposium on Algorithms and Computation, (ISAAC '20)* [137]. A full version is available on ArXiv [138]. Christian and I proposed to study the parameterized complexity of classic graph problems with respect to the c-closure and the weak γ-closure during a research retreat of the AKT group of TU Berlin 2019 in Prignitz. During this retreat, Tomohiro joined the project and we studied clique relaxations such as s-Plex and problems related to bicliques. Tomohiro provided the results for the algorithms with running time $n^{\mathcal{O}(\sqrt{s})}$ for s-Defective Clique. All other results were jointly discovered by all coauthors. The hardness-result for 2-Club discovered by me is not contained in the conference version. The manuscript was prepared by all coauthors.

Chapter 6 is based on the publication "The Parameterized Complexity of s-Club with Triangle and Seed Constraints" written with Jaroslav Garvardt and Christian Komusiewicz which appeared in the *Proceedings of the 33rd International Workshop on Combinatorial Algorithms, (IWOCA '22)* [88]. The full version is available on ArXiv [89]. Christian proposed to study Vertex Triangle s-Club. During the study of this problem, Jaroslav and me proposed to also introduce and study the problems Edge Triangle s-Club and Seeded s-Club. Jaroslav and myself worked out the details for Vertex Triangle s-Club and Edge Triangle s-Club for $\ell = 1$. I extended these results to general ℓ. All three coauthors developed the dichotomy for Seeded 2-Club. Furthermore, I generalized some of these results to general s. Also, I prepared the draft of the manuscript.

Chapter 7 is based on the paper "Efficient Branch-and-Bound Algorithms for Finding 2-Clubs with Triangle Constraints" written with Niels Grüttemeier, Philipp Heinrich Keßler and Christian Komusiewicz which is available on ArXiv [95]. Chris-

tian and I proposed algorithm engineering for Vertex Triangle 2-Club and $\ell = 1$ as the topic for Philipp's bachelor thesis [129]. Afterwards, we decided to extend this implementation to Edge Triangle 2-Club and to general ℓ. For this, Philipp worked as a research assistant in our group. Christian, Niels, and I discovered several speed-ups for the implementation and showed their correctness. Philipp provided the implementation. Philipp and I evaluated our experiments. Christian, Niels, and I prepared the draft.

Chapter 8 is based on the publication "Covering Many (or Few) Edges with k Vertices in Sparse Graphs" written with Tomohiro Koana, Christian Komusiewicz, and André Nichterlein which appeared in the *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science, (STACS '22)* [135]. A full version is available on ArXiv [136]. During our other works on problems parameterized by c-closure and weak γ-closure Tomohiro proposed to study the parameterized complexity of Partial Vertex Cover with respect to c. Later, Tomohiro found the problem Max α-FCGP [29] generalizing Partial Vertex Cover. We decided to study this generalized problem. The details of the annotated version and the positive results for the maximum degree Δ where developed jointly by all authors. Christian provided the details for the negative results for the maximum degree; André and Christian provided the details for the parameters h-index and vertex cover number. Tomohiro provided the positive results for the c-closure and the degeneracy d and I provided the complementing negative results for both parameters. The manuscript was written jointly by all coauthors.

Acknowledgements. I want to thank Christian Komusiewicz for giving me to opportunity to work as a PhD student in his group. Furthermore, I want to thank Christian for proposing interesting problems to study, and that his door was always open for endless discussions with me. Also, I really enjoyed the productive and pleasant atmosphere with my colleagues and friends (in lexicographic order) Jaroslav Garvardt, Niels Grüttemeier, and Nils Morawietz and I am grateful for the time we spend together. Furthermore, I want to thank all of my co-authors (in lexicographic order): Emmanuel Arrighi, Markus Brill, Nils Jakob Eckstein, Piotr Faliszewski, Jaroslav Garvardt, Niels Grüttemeier, Philipp Heinrich Keßler, Tomohiro Koana, Christian Komusiewicz, Nils Morawietz, André Nichterlein, Jannik Schestag, Nimrod Talmon, and Petra Wolf. I am also grateful for the financial support by the Deutsche Forschungsgemeinschaft projects Multivariate Algorithmics for Graph and String Problems in Bioinformatics (MAGZ) and Algorithms for Group Centrality (EAGR) during my PhD. Finally, I want to express my gratitude for my parents Mandy Sommer and Ralf Sommer for all their love and support throughout the years.

Abstract

We study various subgraph problems with applications for example in community detection. In these applications vertices represent agents in a social network or genes in a biological network, and edges represent interactions of the agents or genes, respectively. All of the problems studied in this thesis fit into one of two categories: finding one subgraph fulfilling some specific property, or enumerating all subgraphs of a certain type. We study these problems both theoretically, for example with respect to their classic- and parameterized complexity, and practically, by providing efficient implementations. We study three types of problems:

In many applications like the detection of network motifs, it is important to enumerate all connected induced subgraphs. We compare several implementations for this task and improve upon the best delay due to Elbassioni (JGAA '15). Then, we use the fastest algorithms for this task as a baseline for an exact solver for Connected Fixed Cardinality Optimization. In this problem, one aims to find a set of k vertices maximizing an objective function. We provide several generic pruning rules to speed-up our solver and show for eight example problems that our approach outperforms standard Integer Linear Programs (ILPs) for small values of k.

Community detection is an important task in the analysis of networks. Clique relaxations are a popular tool to model communities. We study the parameterized complexity of finding or enumerating different clique relaxations with respect to the (weak) closure. These parameters were recently discovered by Fox et al. (SIAM J. Comput. '20) and are observed to be small in social networks. Then, we study the diameter-based clique relaxation 2-Club with triangle or seed constraints. For the variants with triangle constraints we provide a dichotomy into cases which admit an FPT-algorithm and those which are W[1]-hard for k. Next, we provide a branch-and-bound algorithm which outperforms an ILP by Almeida and Brás (Comput. Oper. Res. '19). For the seeded variant, we identify several cases which admit an FPT-algorithm and cases which are W[1]-hard for k depending on the structure of the seed.

Finally, we also investigate a local graph partitioning problem generalizing many classic graph problems like Densest k-Subgraph, Maximum Partial Vertex Cover (MaxPVC) and Maximum $(k, n-k)$-Cut. We study the parameterized complexity of this generic graph problem with respect to the solution size k plus one additional structural graph parameter like the maximum degree or the c-closure. We show, for example, that MaxPVC and Maximum $(k, n-k)$-Cut not only behave similarly in terms of fixed-parameter tractability, but also the techniques to obtain them are similar. One of our results is a so-called kernel of size $k^{\mathcal{O}(c)}$ for MaxPVC, thereby answering an open question of Kanesh et al. (STACS '22).

Zusammenfassung

Wir untersuchen verschiedene Subgraphprobleme mit Anwendungen etwa im Finden von Communities. In diesen Anwendungen repräsentieren Knoten Akteure in sozialen Netzwerken oder Gene in einem biologischem Netzwerk und Kanten repräsentieren Interaktionen der Akteure beziehungsweise der Gene. Alle Probleme, welche in dieser Dissertation untersucht werden, gehören zu zwei Kategorien: finden eines Subgraphen, welcher eine bestimmte Eigenschaft erfüllt, oder Aufzählen aller Subgraphen eines bestimmten Typs. Wir untersuchen diese Probleme sowohl theoretisch, zum Beispiel bezüglich deren klassischer oder parametrisierten Komplexität, als auch praktisch, indem wir effiziente Implementierungen entwickeln.

In vielen Anwendungen, wie zum Beispiel dem Finden von Netzwerkmotiven, ist es wichtig alle zusammenhängenden induzierten Subgraphen aufzuzählen. Wir vergleichen verschiedene Implementierungen für diese Aufgabe und verbessern den besten Delay für dieses Problem, welcher von Elbassioni (JGAA '15) nachgewisen wurde. Danach nutzen wir die schnellsten Algorithmen für diese Aufgabe als Grundlage für einen exakten Solver für Connected Fixed Cardinality Optimization. In diesem Problem möchte man eine Menge von k Knoten finden, welche eine Zielfunktion maximiert. Wir stellen viele generische Pruningregeln zur Beschleunigung unseres Solvers zur Verfügung und zeigen für acht Beispielprobleme, dass unser Ansatz für kleine Werte von k schneller ist als Standardformulierungen von Ganzzahligen Linearen Programmen (ILPs).

Das Finden von Communities ist eine wichtige Aufgabe in der Analyse von Netzwerken. Cliquerelaxierungen sind ein weit verbreiteter Ansatz um Communities zu modellieren. Wir untersuchen die parametrisierte Komplexität bezüglich der (weak) closure, um verschiedene Cliquerelaxierungen zu finden oder aufzuzählen. Diese Parameter wurden kürzlich von Fox et al. (SIAM J. Comput. '20) entdeckt und sind klein in sozialen Netzwerken. Danach untersuchen wir die durchmesserbasierte Cliquerelaxierung 2-Club mit Dreiecks- oder Seedconstraint. Für die Varianten mit Dreiecksconstraints beweisen wir eine Dichotomie in Fälle, welche einen FPT-Algorithmus für k zulassen, und solche, welche W[1]-schwer für k sind. Außerdem entwickeln wir einen Branch-and-Bound Algorithmus, welcher schneller ist als ein ILP von Almeida and Brás (Comput. Oper. Res. 2019). Für die Variante mit Seedconstraint finden wir viele Fälle, welche einen FPT-Algorithmus für k zulassen und solche, welche W[1]-schwer für k sind. Diese Klassifikation hängt von der Struktur des Seeds ab.

Schließlich untersuchen wir ein lokales Graphpartitionierungsproblem, welches viele klassische Graphprobleme wie Densest k-Subgraph, Maximum Partial

Vertex Cover (MaxPVC) und Maximum $(k, n-k)$-Cut verallgemeinert. Wir untersuchen die parametrisierte Komplexität von diesem generischen Graphproblem bezüglich der Lösungsgröße k sowie einem zusätzlichen strukturellen Graphparameter wie dem Maximalgrad oder der c-Closure. Zum Beispiel zeigen wir, dass sich MaxPVC und Maximum $(k, n-k)$-Cut nicht nur gleich verhalten im Sinne von FPT-Algorithmen, sondern auch die Techniken, um diese Ergebnisse zu erzielen, identisch sind. Eines unserer Ergebnisse ist ein sogenannter Kern der Größe $k^{\mathcal{O}(c)}$ für MaxPVC. Dieses Resultat beantwortet eine offene Frage von Kanesh et al. (STACS '22).

Contents

Chapter 1

Introduction

Network analysis has a huge variety of applications. In this thesis, we briefly mention three of these applications. For example, it is essential to identify important actors in a social network. One such application is the selection of influencers [117]: A company may have money to pay up to k persons which can advertise their product. Now, the task is to determine k persons such that the number of persons noting the advertisement is maximized. In other words, one searches for k persons with many connections to other persons in the network. Another essential task in social network analysis is the detection of communities [81, 83, 223]. Here, a community is a set of persons with many relations between each other. For example, one may ask what is the biggest community which contains a specific person. Community detection is also essential in computational finances [26, 27, 120] and bioinformatics [211, 215, 232]. A third application is the detection of good teams of k persons [154]: A project requires several skills to be completed. Each person may have a set of skills necessary to complete a project. Furthermore, a set of persons S may form a *team* each team member can work together with each other team member. Now, the question is whether there exists a team of k persons fulfilling all skills necessary for the project. Network analysis is not only important in social science. For example, also in biology network analysis [103, 123, 132] is important.

All these applications have the common property of being NP-hard [80, 87, 126, 154]. Hence, it is very likely that these problems can only be solved in exponential time in the input size. From a theoretic perspective, often the *parameterized* complexity of such problems is studied [29, 70, 143, 205], and from a practical perspective Integer Linear Programs (ILPs) [14, 185, 203] or branch-and-bound algorithms are provided [43, 109, 145].

In this thesis we continue both of these research directions: In the theoretical

part, we study the (parameterized) complexity of various subgraph problems. We provide both positive and the negative results. The positive results are shown by concrete algorithms and the negative results are based on standard, widely believed complexity assumptions. Furthermore, we show that some of the algorithmic ideas lead to fast implementations which beat the state of the art in terms of running time. In the following, we discuss three different types of problems which we study in this thesis.

1.1 Connected Subgraph Problems

One very important property is connectivity: A standard concept of being a community is that all its members can interact with each other. In other words, two persons cannot be part of a community if they do not interact. Interactions are modelled via connectivity. Hence, it is an important task to find connected communities or groups (induced connected subgraphs) [39, 127, 227]. Connectivity is also important for reachability [48, 105]. Many of these reachability problems are special cases of Connected Fixed-Cardinality Optimization (CFCO) [39, 205]. In this generic problem one searches for a connected vertex set of size k maximizing an objective function. One example is Connected Densest k Subgraph [39]. In this problem one searches for connected vertex set S of size k maximizing the number of edges having both endpoints in S. Another application in which connected groups are important is in the identification of statistically overrepresented induced subgraphs of small size, so-called network motifs [127, 227].

In Chapter 3, we study the task of enumerating all connected induced subgraphs of size exactly k. This task is essential for the above-mentioned problems relying on connectivity, like the identification of network motifs [127, 227]. The running time of algorithms for this task is linear in the input size and the number of connected induced subgraphs of size exactly k [150]. But even in very sparse graphs, the number of connected induced subgraphs of size exactly k is exponentially large [28]. Thus, also the running time of such algorithms is exponential in the input size. Hence, to judge the effectiveness of such algorithms, one studies the *delay*, the time spend between two consecutive outputs. In Chapter 3, we improve upon the best known delay due to Elbassioni [67] for enumerating all connected induced subgraphs of size exactly k. Furthermore, we perform extensive experiments to evaluate the performance of our algorithms and existing algorithms for this task. Also, we study related enumeration problems; for example the enumeration of all connected induced subgraphs of size *at most* k.

In Chapter 4, we present a generic solver for CFCO. As noted above, many

classic graph problems are special cases of CFCO [39, 205]. These special cases have applications in the design of wildlife corridors [48] and in oil field leasing [105]. Currently, the state of the art for such problems relies on ILPs [47, 197]. A different approach, followed by Komusiewicz et al. [150, 151] for a problem called μ-Clique, relies on the enumeration of all connected induced subgraphs of size exactly k, evaluating the objective function for this problem for each induced subgraph of size k, and providing some problem-specific pruning rules. In Chapter 4, we lift this approach to a generic solver of CFCO with pruning rules applying to large subclasses of CFCO. In our framework, a user only has to implement the objective function and check whether our generic pruning rules apply for his specific problem. We were able to show that our algorithm outperforms standard ILP formulations for eight example problems for k up to 20 on real-world instances with up to 500 000 vertices.

1.2 Community Detection

As discussed above, community detection is essential in many domains [27, 211, 223]. Informally, the following properties are highly desirable for a community, for example a social group [40, 223]: First, each member of the community should know many other members of the community, that is, the degree of each vertex in the community should be high. Second, many members of the community should know each other, that is, there should be many edges within the community. Third, the members of the community should know each other directly, or over at most z other members of the community. In other words, the diameter of the community should be small. The most strict model for all these three properties are cliques.

Definition 1.1. A vertex set S is a *clique* if each two distinct vertices $u, w \in S$ are adjacent.

A clique fulfills all of the three informal properties above perfectly: Each vertex has degree $|S| - 1$, the number of edges with both endpoints in S is $\binom{|S|}{2}$, and the diameter of $G[S]$, the subgraph induced by S, is one. The clique model leads to the following natural problem in which we aim to find a clique of size at least k.

Clique
Input: An undirected graph $G = (V, E)$ and an integer k.
Question: Does G contain a clique of size k?

Clique is NP-hard [87, 126]. and has applications in computational finances [26, 115], identification of web communities [81] computational biochemistry [57], and bioinformatics [215].

In many applications the clique model, however, is too strict. In other words, less strict definitions of being a community are desirable for example in social networks [21]. One main reasons for this is noise in the input data [7]: The edges in a graph represent, for example, relations between persons. These relations are observed by experiments and thus some relations can be overseen by these experiments. Consequently, two vertices may be non-adjacent despite being in a community. In consequence, to overcome these issues, a community model should allow for some non-edges.

To overcome the above-mentioned issues of the clique model, one popular approach is to consider *clique relaxations*. As the name suggests in these models [168, 191, 193] at least one clique-defining property is relaxed, for example the diameter. Because of the importance of clique relaxations there is a large number of works considering such problems. For an overview on different clique relaxations we refer to the surveys of McClosky [168] and of Pattillo et al. [193]. For an overview on results of the computational complexity and results on the approximation hardness we refer to the survey of Balasundaram and Pajouh [15]. Pattillo et al. [192] provided a historical overview in which chronological ordering these concepts and results have been discovered. Furthermore, Komusiewicz [143] provided a survey about the parameterized complexity for clique relaxations. In the following, we present some clique relaxation, which will be studied in further detail in this thesis.

One option to relax the clique model is to relax the vertex-degree. Clearly, a vertex set S is a clique if every vertex in S has exactly $|S| - 1$ neighbors in S. This property can be relaxed as follows:

Definition 1.2. A vertex set S is an *s-plex* if every vertex in S has at least $|S| - s$ neighbors in S.

Observe that cliques are 1-plexes. The s-plex model was introduced by Seidman and Foster [208]. This definition leads to the following problem in which one aims to detect an s-plex of size at least k.

s-Plex
Input: A graph G and an integer k.
Question: Does G contain an s-plex of size at least k?

The NP-hardness of s-Plex follows from a result of Lewis and Yannakakis [156]. The s-plex model has many applications, for example in finding profitable diversified portfolios on the stock market [27], in the analysis of social networks [13], and in graph-based text mining [16].

Another option to relax the clique model is to relax the total number of edges within the community. Clearly, a vertex set S forms a clique if the subgraph induced by S has exactly $\binom{|S|}{2}$ edges. This property can be relaxed as follows:

Definition 1.3. A vertex set S is an *s-defective clique* if the number of edges in the subgraph induced by S is at least $\binom{|S|}{2} - s$.

Observe that cliques are 0-defective cliques and each s-defective clique is also an s-plex. This model was introduced by Yu et al. [232]. This definition leads to the following decision problem:

s-Defective Clique
Input: A graph G and an integer k.
Question: Does G contain an s-defective clique of size at least k?

Similar to s-Plex, the NP-hardness of s-Defective Clique follows from a result of Lewis and Yannakakis [156]. Furthermore, s-Defective Clique has applications in predicting interactions in protein networks [232], representation of stock mark dynamics [120], and cancer prognosis [201].

Another option to relax the clique model is to allow greater distances within the vertices in a group. Clearly, a vertex set S is a clique if each two distinct vertices in S have distance 1. This property can be relaxed as follows:

Definition 1.4. A set of vertices S of G is an *s-club* if each pair of vertices in S has distance at most s in $G[S]$, the subgraph induced by S.

In other words, S is an s-club if $G[S]$ has diameter at most s. Observe that cliques are 1-clubs. This clique relaxation was proposed by Mokken [170]. This definition leads to the following decision problem:

s-Club
Input: A graph G and an integer k.
Question: Does G contain an s-club of size k?

The NP-hardness of s-Club for $s \geq 2$ was proven by Bourjolly et al. [31]. Furthermore, s-Club has applications in community mining [83], and in the detection of protein complexes [190].

In Chapter 5 we study the clique relaxations s-Plex, s-Defective Clique and s-Club in terms of their parameterized complexity with respect to k plus the (weak) closure. Our aim is to extend previous results on the parameterized complexity for

these three clique relaxations with respect to structural graph parameters like the maximum degree Δ or the degeneracy d (plus the solution size k) [110, 131, 143, 198]. The (weak) closure is a newly discovered graph parameter [84] and bounds the number of common neighbors of non-adjacent vertices. Both parameters are observed to be small in social networks [84] and are motivated by the so-called triadic-closure principle. For a formal definition of these parameters we refer to Chapter 5. For many clique relaxations, for example s-plexes, we show that the enumeration of all maximal vertex sets being this clique relaxation can be done efficiently in weakly closed graphs. But not all clique relaxations behave that nicely: the detection of a 2-club of size at least k remains NP-hard even on graphs with constant closure, as we show.

In Chapter 6, we study three different variants of the s-club model: Vertex Triangle s-Club, Edge Triangle s-Club, and Seeded s-Club. These models are motivated by some undesirable behaviour of the s-club model in real-world instances: The largest s-club is usually very sparse [109]. This is especially true for $s = 2$: often the largest 2-club in a graph is the vertex v of maximum degree together with its neighbors [109]. To overcome this issue, many different augmentations of the s-club model have been studied [41, 145, 193, 222]. In the first two variants (Vertex Triangle s-Club and Edge Triangle s-Club) we additional require that each vertex or edge, respectively, is contained in at least ℓ triangles. Here, ℓ is a parameter which is part of the input. In the third variant we require that a fixed set of vertices, denoted as *seed*, is contained in the solution. The Vertex Triangle s-Club problem was studied in the literature before [3, 41], and the other two problems are introduced by us. The variant with seeds can be used in community detection, where we are often interested in finding communities containing some set of fixed vertices [124, 230]. In Chapter 6, we study the classic and the parameterized complexity of these problems with respect to the number of triangles or the structure of the seed, respectively. We show, for example, that in contrast to s-Club all three problems do not admit in all cases an FPT-algorithm with respect to k.

In Chapter 7 we present an exact branch-and-bound algorithm for Vertex Triangle 2-Club and Edge Triangle 2-Club. Our work builds on ILPs [14, 185, 203], and branch-and-bound algorithms [43, 109] for 2-Club We provide new reduction rules and one new lower bound for both problems. Furthermore, we adapt techniques to find lower bounds from an existing exact solver for a 2-Club variant by Komusiewicz et al. [145] for the two above mentioned problems. We evaluated our implementations for Vertex Triangle 2-Club and Edge Triangle 2-Club on a large number of real-world instances. Furthermore, we showed that our implementation outperforms an existing ILP for Vertex Triangle 2-Club [3].

1.3 Subgraph Problems Depending on the Entire Graph

In Chapter 8, we study the parameterized complexity of a problem generalizing the NP-hard problems Maximum Partial Vertex Cover, and Max $(k, n-k)$-Cut. In all models and problems discussed so far, the property such as being connected, being a clique or being an s-club only depends on the structure of $G[S]$ where S is the vertex set of the solution. In many problems, like Maximum Partial Vertex Cover (see discussion below), this is not sufficient, that is, also the structure of the remaining vertices $V \setminus S$ and the structure of the edges having exactly one endpoint in S is relevant: For example in Maximum Partial Vertex Cover one aims to find a set S of k vertices such that the number of edges having at least one endpoint in S is as large as possible [101, 133]. Another example is Max $(k,n-k)$-Cut where one asks for a set S of k vertices such that the number of edges having exactly one endpoint in S is as large as possible [38, 204]. In Partial Dominating Set one asks for a set S of k vertices such that the size all vertices with distance at most 1 to S is as large as possible [134, 178]. Another example is Group Closeness Centrality, where one searches a set S of k vertices such that the sum of the distances from each vertex in the graph to the set S is minimal [73].

In Chapter 8, we study a local graph partitioning problem generalizing Maximum Partial Vertex Cover, and Max $(k, n-k)$-Cut. The common feature of all these problems is that one takes the sum of the number of edges having both endpoints in S and the number of edges having exactly one endpoint in S multiplied with some problem specific weights into account. We investigate the parameterized complexity of this generic graph problem with respect to the size k of the vertex set S plus one additional structural graph parameter like the maximum degree Δ or the degeneracy d. One surprising result is that we obtain identical results for the problems Maximum Partial Vertex Cover and Max $(k, n-k)$-Cut, for example a tight kernel of size $k^{\mathcal{O}(d)}$, and that these results are obtained by the same techniques.

Chapter 2

Preliminaries

In this chapter, we present our notation and we give an overview of the central techniques and concepts of graph theory and (parameterized) complexity theory that we use in this work.

By $\mathbb{N}$ we denote the set of positive integers and by $\mathbb{N}_0$ we denote the set of non-negative integers. For $p \leq q \in \mathbb{N}$, we write $[p, q]$ for the set $\{p, p+1, \ldots, q\}$ and $[q]$ for $[1, q]$. A *partition* $\mathcal{P} \coloneqq (P_1, \ldots, P_\ell)$ of a set P is a family of pairwise disjoint subsets of P whose union is P. A function f is *bounded* by another function g if $f(x) \in \mathcal{O}(g(x))$.

2.1 Graph Theory Notation

We consider simple and undirected graphs. Next, we provide the main graph-theoretic notation we use throughout this work. For a more detailed introduction into graph theory we refer to the standard textbooks [58, 229].

An *undirected simple graph* is a tuple $G \coloneqq (V, E)$ where V denotes the set of *vertices* and $E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$ denotes the set of *edges*. Furthermore, by $V(G)$ we denote the set of vertices of G and by $E(G)$ we denote the set of edges of G. We let n and m denote the order of G and the number of edges in G, respectively. For simplicity we denote an edge $\{u, v\}$ by uv. Furthermore, u and v are the *endpoints* of uv. The *order* of a graph is the number of its vertices. Two vertices u and v are called adjacent if $uv \in E(G)$. Two edges e_1 and e_2 are *incident* if they have exactly one common endpoint. By $E(V, W) \coloneqq \{vw \in E(G) \mid v \in V \text{ and } w \in W\}$ we denote the set of edges *between* V and W. For simplicity, we use $E(V) \coloneqq E(V, V)$.

A graph $G' \coloneqq (V', E')$ is a *subgraph* of G if $V' \subseteq V$, and $E' \subseteq E$. For a vertex set S, by $G[S] \coloneqq (S, \{uv \in E(G) \mid u, v \in S\})$ we denote the *subgraph of* G

induced by S. Similarly, for an edge set F we define the *edge-induced subgraph of* G as the graph $G' \coloneqq (V_F, F)$, where V_F is the set of endpoints of the edges in F. Furthermore, the *size* of the edge-induced subgraph of an edge set F is $|F|$, the number of edges. Let $X, Y \subseteq V(G)$ be vertex subsets. For $X \cap Y = \emptyset$, we use $G[X, Y] \coloneqq (X \cup Y, \{xy \in E(G) \mid x \in X, y \in Y\})$ to denote the *bipartite subgraph of* G *induced* by X, Y. We let $G - X \coloneqq G[V \setminus S]$ denote the subgraph obtained by *removing* the vertices in X.

A set of vertices $\{v_i \mid i \in [p]\}$ is a *non-induced path* if $v_i v_{i+1} \in E(G)$ for each $i \in [p-1]$. Its *length* is $p-1$. Furthermore, a vertex set $S = \{v_i \mid i \in [p]\}$ is an *induced path* if $E(G[S]) = \{v_i v_{i+1} \mid i \in [p-1]\}$. In the following, an induced path with p vertices is denoted by P_p and simply called *path*. Similar, a set of vertices $\{v_i \mid i \in [p]\}$ is a *cycle* C_p if $E(G[S]) = \{v_i v_{i+1} \mid i \in [p-1]\} \cup \{v_p v_1\}$ The length of C_p is p. The distance $\mathrm{dist}_G(u, v)$ between two vertices u and v in a graph G is the length of a shortest path between u and v. Furthermore, for a set $W \subseteq V(G)$ we define $\mathrm{dist}_G(u, W) \coloneqq \min_{w \in W} \mathrm{dist}(u, w)$. Hence, two vertices in C_p have distance at most $\lfloor p/2 \rfloor$. We denote by $\mathrm{diam}_G(G) \coloneqq \max_{u,v \in V(G)} \mathrm{dist}_G(u, v)$ the *diameter* of G.

Let $S \subseteq V(G)$ be a vertex set. We denote by $N_i(S) \coloneqq \bigcup_{w \in S} N_i(w) \setminus (\bigcup_{j<i} N_j(w) \cup S)$ the *open* i*-neighborhood* of S and by $N_i[S] \coloneqq \bigcup_{j \leq i} N_i(S) \cup S$ the *closed* i*-neighborhood* of S. For simplicity, by $N(S) \coloneqq N_1(S)$ we denote the *open neighbors* and by $N[S] \coloneqq N_1[S]$ the *closed neighbors* of S. For simplicity, we use $N_i(v) \coloneqq N_i(\{v\})$ and $N_i[v] \coloneqq N_i[\{v\}]$ We also use the notation $N_G^{\cap}(X) \coloneqq \bigcap_{x \in X} N(x)$ to denote the common neighbors of a vertex set X.

If X is a singleton $\{x\}$ we may write $G - x$ instead of $G - \{x\}$. Let $v \in V(G)$. We denote the *degree* of v by $\deg_G(v) \coloneqq |N(v)|$. We call v *isolated* if $\deg_G(v) = 0$ and *non-isolated* otherwise. We also say that v is a *leaf vertex* if $\deg_G(v) = 1$ and a *non-leaf vertex* if $\deg_G(v) \geq 2$. Moreover, we say that v is *simplicial* if $N(v)$ is a clique. The *maximum* and *minimum degree* of G are $\Delta_G \coloneqq \max_{v \in V(G)} \deg_G(v)$ and $\delta_G \coloneqq \min_{v \in V(G)} \deg_G(v)$, respectively. The *degeneracy* of G is $d_G \coloneqq \max_{S \subseteq V(G)} \delta_{G[S]}$. The h_G*-index* of a graph G is the largest integer h such that G has at least h vertices of degree at least h [71]. The *closure number* $\mathrm{cl}_G(v)$ of v is $\max_{u \in V(G) \setminus N[v]} |N(v) \cap N(u)|$. We say that G is c*-closed* if $\mathrm{cl}_G(v) < c$ for each vertex $v \in V(G)$. Furthermore, the *closure number* of a graph G is the smallest integer c such that G is c-closed [84]. We say that G is weakly γ-closed if every induced subgraph G' of G has a vertex $v \in V(G')$ such that $\mathrm{cl}_{G'}(v) < \gamma$. Furthermore, the *weak closure number* of a graph G is the smallest integer γ such that G is weakly γ-closed [84]. We denote the size of a *smallest vertex cover* (a set of vertices that covers all edges) of a graph G by vc_G. We drop the subscript $\cdot_G$ when it is clear from context.

A graph G has *girth* g if the shortest cycle in G has length g. Two vertices u and v

are *connected* if there exists a path P such that $u, v \in P$. A *connected component* of G is a maximal subgraph where any two vertices are connected to each other. Furthermore, a graph $G = (V, E)$ is called *k-edge-connected* if $G - F$ is connected for each $F \subseteq E$ of size at most k. Also, $G = (V, E)$ is called *k-vertex-connected* if $G - F$ is connected for each $F \subseteq V$ of size at most k. Observe that a graph G is 1-edge-connected if and only if G is 1-vertex-connected if and only if G has exactly one connected component. An edge e is a *bridge* if $G - e$ has one more connected component than G. A graph G is *complete* if any two vertices of G are adjacent and a vertex set S is a *clique* if each two distinct vertices in S are adjacent. A clique consisting of three vertices is referred to as a *triangle*. A graph G is *edgeless* if any two vertices of G are non-adjacent and a vertex set S is an *independent set* if each two distinct vertices in S are non-adjacent. Two graphs $G \coloneqq (V_1, E_1)$ and $H \coloneqq (V_2, E_2)$ are *isomorphic* if there exists a bijective function $f : V_1 \mapsto V_2$ such that $uv \in E_1$ if and only if $f(u)f(v) \in E_2$ for each $u, v \in V_1$. Furthermore, a graph G is called *H-free* if G does not contain an induced subgraph isomorphic to H. A graph is *r-regular* if every vertex has degree r.

Ramsey numbers. Ramsey's theorem states that for every $p, q \in \mathbb{N}$, there exists an integer $R(p, q)$ such that any graph on at least $R(p, q)$ vertices contains either a clique of size p or an independent set of size q. The numbers $R(p, q)$ are referred to as *Ramsey numbers*. Although the precise values of Ramsey numbers are not known, some upper bounds have been proven. For instance, it holds that $R(p, q) \leq \binom{p+q-2}{p-1}$ [122]. The proof for this upper bound is constructive. More precisely, given a graph G on at least $\binom{p+q-2}{p-1}$ vertices, we can find in time $n^{\mathcal{O}(1)}$ either a clique of size p or an independent set of size q.

2.2 Classic Computational Complexity

Now, we provide the main computational complexity theoretic tools. For more details on the theory of NP-hardness and NP-completeness we refer to the standard textbooks [9, 87, 188].

Up to now, we only considered decision problems. Nonetheless, the algorithms we describe in this work are also capable of solving the natural optimization problem that corresponds to the decision problem. Usually this is accompanied with a small running time overhead.

Many of the computational problems in this work are formulated as decision problems. Formally, a *decision problem* is defined as a language $L \subseteq \Sigma^*$ where Σ is

a finite alphabet. The question is whether for a given input $x \in \Sigma^*$ we have $x \in L$ or $x \notin L$. An instance x such that $x \in L$ is called *yes-instance* and an instance x such that $x \notin L$ is called a *no-instance*. In this work, we only deal with *decidable* problems. These are problems for which there exists an algorithm that terminates for each input.

We focus on the amount of computational resources which are necessary to solve a problem. These resources are called complexity measures. Here, we only focus on the standard complexity measures *running time*, that is, the time needed to terminate on the input, and *space*, that is, the maximal space used by the algorithm. Given an algorithm A that decides if $x \in L$, the running time and the space of A is measured as a function of the input size $|x|$ for all $x \in \Sigma^*$. In the following, we say that an algorithm needs *polynomial space*, if the total space used by the algorithm is poly($|x|$). Furthermore, if the space usage is linear in $|x|$, we say that the algorithm needs *linear space*.

The two most prominent classes of decidable decision problems are P and NP. The class P contains all problems that can be decided in polynomial time by a deterministic Turing machine and the class NP contains all problems that can be decided in polynomial time by a non-deterministic Turing machine. It is widely believed that P $\neq$ NP. In other words, it is widely believed that there are problems in NP that can *not* be solved by a deterministic Turing machine in polynomial time.

A very important complexity class in this context is the class of NP-hard problems. These are defined by polynomial-time reductions between problems. A problem $A \subseteq \Sigma^*$ *reduces* to a problem $B \subseteq \Sigma^*$ if there is a function $f : \Sigma^* \mapsto \Sigma^*$ which can be computed in polynomial-time such that $x \in A$ if and only if $f(x) \in B$. The existence of such a reduction is abbreviated as $A \leq_p B$. In other words, problem B is at least as hard as problem A with respect to polynomial-time solvability. Now, a problem B is NP-hard if $A \leq_p B$ for all problems $A \in$ NP. Hence, an NP-hard problem is at least as hard as any problem in NP with respect to polynomial-time solvability. Furthermore, an NP-hard problem that is also contained in NP is called NP-complete. A prominent NP-complete problem is Clique.

Clique

Input: An undirected graph $G = (V, E)$ and an integer k.
Question: Does G contain a clique of size at least k?

This is a hint that P is not equal to NP.

2.3 Parameterized Complexity

In this thesis, one of our main objectives is the study of the parameterized complexity of NP-hard graph problems. Next, we describe the main definitions and the most common techniques of parameterized complexity. For a more detailed introduction we refer to the standard textbooks [53, 63, 82, 183].

Fixed-Parameter Tractability. The main idea of fixed-parameter algorithms is to allow for a super-polynomial running time as long as it only depends on a *parameter* k. Hence, parameterized decision problems consist of two components, the language and the parameter.

Definition 2.1. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}_0$, where Σ is a finite alphabet. The first component is referred to as the *input* and the second component is referred to as the *parameter*.

In the following, we call an instance $(x, k) \in \Sigma^* \times \mathbb{N}_0$ a *yes-instance* of L if $(x, k) \in L$, and otherwise, (x, k) is called a *no-instance*.

Definition 2.2. A parameterized problem L is *fixed-parameter tractable* if there is a deterministic algorithm that decides in $f(k) \cdot \text{poly}(|x|)$ time for every input instance (x, k) of L whether $(x, k) \in L$. Here, f is a computable function only depending on k. The complexity class of all parameterized problems that are fixed-parameter tractable is called *FPT*.

Next, we define the class of problems which are polynomial-time solvable for constant k.

Definition 2.3. A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}_0$ is called *slice-wise polynomial* if there exist a computable function $g : \mathbb{N}_0 \mapsto \mathbb{N}_0$, and an algorithm that decides whether $(x, k) \in L$ in $|(x, k)|^{g(k)}$ time for a given instance (x, k). The complexity class of all parameterized problems that are slice-wise polynomial is called *XP*.

Since each fixed-parameter tractable algorithm is also slice-wise polynomial, the class FPT is contained in the class XP.

The W-Hierarchy. If P $\neq$ NP and all parameterized problems are fixed-parameter tractable, then parameterizing any NP-hard problem by any constant parameter would result in a deterministic algorithm with a polynomial running-time. Since it is widely believed that P $\neq$ NP it is thus also widely assumed that not each

parameterized problem is contained in FPT. More interestingly, there are many slice-wise polynomial problems for which it is still unknown whether they are also fixed-parameter tractable. Downey and Fellows developed the so-called *W-Hierarchy* to show that some problems in XP are unlikely to be in FPT [61, 62, 63]. Downey and Fellows introduced a complexity class W[i] for each integer i. It is widely believed that FPT $\subsetneq$ W[1] $\subsetneq$ W[2] $\subsetneq \ldots \subsetneq$ XP. To show that a problem is hard for for a class W[i] for some integer i, Downey and Fellows introduced a parameterized variant of reductions.

Definition 2.4. Let L_1 and L_2 be two parameterized problems and let f and g be two computable functions only depending on k. A *parameterized reduction* from L_1 to L_2 is an algorithm that, for each instance (x, k) of L_1 computes in $f(k) \cdot \text{poly}(|x|)$ time an instance (x', k') of L_2 such that $k' \leq g(k)$ and (x, k) is a yes-instance if and only if (x', k') is a yes-instance.

Let L be a parameterized problem and let L' be another parameterized problem which is W[i]-hard such that there exists a parameterized reduction from L' to L. Then, L is also hard for the class W[i]. Furthermore, a parameterized problem is W[i]-*complete* if it is W[i]-hard, and contained in W[i]. It is widely assumed that a W[i]-hard problem for any integer i is not fixed-parameter tractable.

Many of our hardness results are shown by a reduction from Clique, which is known to be W[1]-hard with respect to the standard parameter solution size k [53, 63].

Data Reduction and Problem Kernelization. A general technique which is often used in practice to solve NP-hard problems is polynomial-time preprocessing. In general, the hope is that after the preprocessing the instance is "smaller" and can then be solved in super-polynomial time. In classic complexity theory there is no measurement of the effect of the preprocessing since any algorithm that reduces an instance x of a NP-hard problem to an instance x' of the same problem in polynomial time such that $|x| > |x'|$ implies a polynomial-time algorithm for this problem since this algorithm may be repeat this preprocessing $|x|$ times until an instance of constant size remains.

For parameterized problems, the existence of the parameter allows us to be capture this notion. The basic idea is that if the instance is "too big" in terms of k, then the preprocessing can be applied and decreases the size of the instance. This idea is captured in the following definition.

Definition 2.5. Let L be a parameterized problem and let (x, k) and (x', k') be two instances of L. A reduction to a *(problem) kernel* for L is a polynomial-time algorithm that computes an instance (x', k') such that

1. $k' + |x'| \leq g(k)$ for some computable function g only depending on k, and

2. (x, k) is a yes-instance of L if and only if (x', k') is a yes-instance of L.

The instance (x', k') obtained by the kernelization algorithm is referred to as the *problem kernel*. Furthermore, if g is a polynomial, we say that the kernel is a *polynomial problem kernel*. Also, there is a nice relation between problems which are fixed-parameter tractable and problems which have a kernel.

Theorem 2.6 ([53])**.** *Let L be a parameterized problem. The problem L is fixed-parameter tractable if and only if L admits a problem kernel.*

To obtain problem kernels often *data reduction rules* are used. A data reduction rule is an algorithm that transforms an instance (x, k) of a parameterized problem L into an instance (x', k') of L. A reduction rule is *correct* if (x, k) is a yes-instance of L if and only if (x', k') is a yes-instance of L. A data reduction rule has been *exhaustively applied* if any further application of the data reduction rule on the instance (x, k) results in the same instance (x, k). Furthermore, an instance (x, k) of a parameterized problem is *reduced* with respect to a set of reduction rules if the rules have been exhaustively applied on the instance (x, k).

Despite every fixed-parameter tractable problem admitting a problem kernel, there exist problems which are unlikely to admit a polynomial problem kernel [53]. All results showing no polynomial kernel for a problem admitting an FPT-algorithm are based on the assumption that coNP $\not\subseteq$ NP/poly. The fact that a parameterized problem does not admit a polynomial problem kernel can be transferred to other problems with the following reduction.

Definition 2.7. Let L_1 and L_2 be two parameterized problems and let p be a polynomial. A *polynomial parameter transformation (PPT)* is an algorithm that maps in polynomial time an instance (x, k) of L_1 to an instance (x', k') of L_2 such that

1. (x, k) is a yes-instance of L_1 if and only if (x', k') is a yes-instance of L_2, and

2. $k' \leq p(k)$.

Now, if L_1 does not admit a polynomial kernelization unless coNP $\subseteq$ NP/poly and there is a PPT from L_1 to L_2, then L_2 does not admit a polynomial kernelization unless coNP $\subseteq$ NP/poly. Another way to exclude polynomial kernels under the assumption coNP $\not\subseteq$ NP/poly relies on cross-compositions. For this, the following definition is necessary.

Definition 2.8. An equivalence relation R on Σ^* is called a *polynomial equivalence relation* if the following two conditions hold:

1. There is an algorithm that given two strings $x, y \in \Sigma^*$ decides whether x and y belong to the same equivalence class in $(|x| + |y|)^{\mathcal{O}(1)}$ time, and

2. for any finite set $S \subseteq \Sigma^*$ the equivalence relation R partitions the elements of S into at most $(\max_{x \in S} |x|)^{\mathcal{O}(1)}$ classes.

Now, we can define cross-compositions.

Definition 2.9 ([25])**.** Let L_1 be a language and let L_2 be a parameterized problem. The problem L_1 *cross-composes* into L_2 if there is a polynomial equivalence relation R and an algorithm which, given 2^t strings $x_1, x_2, \ldots, x_{2^t}$ belonging to the same equivalence class of R, computes an instance $(x^*, k^*) \in \Sigma^* \times \mathbb{N}$ in time polynomial in $\sum_{i=1}^{2^t} |x_i|$ such that:

1. $(x^*, k^*) \in L_2$ if and only if $x_i \in L_1$ for some $i \in [2^t]$, and

2. k^* is bounded by a polynomial in $\max_{i=1}^{2^t} |x_i| + t$.

The tools presented so far only show super-polynomial lower-bounds under the standard assumption coNP $\not\subseteq$ NP/poly. Next, a more refined technique is presented. This approach allows showing polynomial lower-bounds under the standard assumption coNP $\not\subseteq$ NP/poly. First, we extend PPTs to also show polynomial lower bounds.

Definition 2.10 ([112])**.** Let L_1 and L_2 be two parameterized problems. A *linear parameter reduction* is an algorithm that maps an instance (x, k) of L_1 to an instance (x', k') of L_2 such that

1. (x, k) is a yes-instance of L_1 if and only if (x', k') is a yes-instance of L_2, and

2. $k' \in \mathcal{O}(k)$.

Linear parameter reductions can be used as follows.

Lemma 2.11 ([112])**.** *Let L_1 and L_2 be two parameterized problems and let q be a fixed integer. If L_1 admits a linear parameter reduction to L_2 and L_1 has a kernel of size $\mathcal{O}(k^d)$, then L_2 also has a kernel of size $\mathcal{O}(k^d)$.*

Second, we lift cross-compositions to show polynomial lower bounds.

Definition 2.12 ([56, 112])**.** Let $q \geq 1$ be an integer, let $L_1 \subseteq \{0,1\}^*$ be a decision problem, and let $L_2 \subseteq \{0,1\}^* \times \mathbb{N}$ be a parameterized problem. A *weak q-composition* from L_1 to L_2 is a polynomial time algorithm that on input $x_1, \ldots, x_{t^q} \in \{0,1\}^n$ outputs an instance $(x', k') \in \{0,1\}^* \times \mathbb{N}$ such that:

1. $(x', k') \in L_2 \Leftrightarrow x_i \in L_1$ for some $i \in [t^q]$, and
2. $k' \leq t \cdot n^{\mathcal{O}(1)}$.

Now, the existence of a weak q-composition allows us to show a polynomial lower bound. These lower bounds hold for *compressions*.

Definition 2.13. A *polynomial compression* of a parameterized language $L \subseteq \Sigma^* \times \mathbb{N}$ into an language $L' \subseteq \Sigma^*$ is an algorithm that takes as input an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ and returns a string y in time polynomial in $|x| + k$ such that

1. $|y| \leq P(k)$ for some polynomial p, and
2. $y \in L'$ if and only if $(x, k) \in L$.

Lemma 2.14 ([53, 56, 112])**.** *Let $q \geq 1$ be an integer, let $L_1 \subseteq \{0,1\}^*$ be an NP-hard problem, and let $L_2 \subseteq \{0,1\}^* \times \mathbb{N}$ be a parameterized problem. If there is a weak q-composition from L_1 to L_2, then L_2 has no compression of size $\mathcal{O}(k^{q-\epsilon})$ for any $\epsilon > 0$, unless* coNP $\subseteq$ NP/poly.

Relations between Parameters. Because of relations between parameters some results transfer to different parameters. Let x be the input of a parameterized problem and let k and ℓ be two parameters corresponding to the input. We say that parameter k is *bounded* by parameter ℓ if $f : (x, k) \mapsto k$ is bounded by $g : (x, \ell) \mapsto \ell$. Recall, that a function f is *bounded* by another function g if $f(x) \in \mathcal{O}(g(x))$. Now, a "positive" result for the smaller parameter k transfers to the lager parameter ℓ. For example, if (x, k) admits an FPT-algorithm, then also (x, ℓ) admits an FPT-algorithm. Analogously, "negative" results for the larger parameter ℓ transfer to the smaller parameter k. For example, if (x, ℓ) does not admit a polynomial kernel for parameter ℓ under standard assumptions, then also (x, k) does not admit a polynomial kernel for k under the same assumption.

Turing Kernelization. In kernels we create a single instance. In *Turing Kernels* the algorithm can query an oracle to decide the answer to small instances of a specific problem in constant time. This concept is captured in the following definition.

Definition 2.15. Let L be a parameterized problem and let $f : \mathbb{N} \to \mathbb{N}$ be a computable function. A *Turing kernelization* of *size* f for L is an algorithm that decides whether a given instance $(x, k) \in \Sigma^* \times \mathbb{N}$ is contained in L in time polynomial in $|x| + k$, when given access to an oracle that decides membership in L for any instance (x', k') with $|x'| + k' \leq f(k)$ in a single step.

2.4 Concepts for Enumeration Problems

In this thesis, we also study enumeration problems, that is, problems in which all solutions have to be output. For more details on enumeration problems we reefer to the book of Marino [165] and to the PhD thesis of Strozecki [213]. An *enumeration problem* is a relation $R \subseteq \Sigma^* \times \Sigma^*$. Let $R(x) \coloneqq \{y \in \Sigma^* \mid (x, y) \in R\}$ be the strings which are *in relation* with x. Note that $R(x)$ is not necessarily finite. A string $y \in R(x)$ is a *solution*. All enumeration problems are special cases of the following problem.

ENUMERATE-R
Input: An instance $x \in \Sigma^*$.
Task: Output $R(x) \coloneqq \{y \in \Sigma^* \mid (x, y) \in R\}$.

Usually, the *total time* to output all solutions is considered. Since the number of solutions might be exponential in the input size, enumeration algorithms often need exponential time. Hence, different to decision problems, also other running-time measures are used for enumeration problems: An algorithm is called *output-polynomial* if its running-time is $(|x| + |R(x)|)^{\mathcal{O}(1)}$. Let $s_1, \ldots, s_{|R(x)|}$ be the ordering of the elements of $R(x)$ enumerated by an algorithm $\mathcal{A}$. We say that $\mathcal{A}$ is *incremental polynomial* if s_i is output in $(|x| + i)^{\mathcal{O}(1)}$ time. The *delay* of $\mathcal{A}$ is the maximum of

1. the time needed to compute s_1,
2. the time needed between the output of s_i and s_{i+1}, and
3. the time needed to terminate after the output of the last solution $s_{|R(x)|}$.

Furthermore, $\mathcal{A}$ is called a *polynomial delay* algorithm, if the delay of $\mathcal{A}$ is $|x|^{\mathcal{O}(1)}$. Note that each polynomial-delay algorithm is also incremental polynomial and that each incremental polynomial algorithm is also output-polynomial.

One general paradigm to obtain a polynomial delay algorithm for an enumeration problem is the *reverse search* framework introduced by Avis and Fukuda [11]. The basic idea is to construct a tree where each node represents a unique solution of the enumeration process. By traversing this tree from the root, each element is enumerated exactly once.

Chapter 3

Enumerating Connected Induced Subgraphs: Improved Delay and Experimental Comparison

The enumeration of connected subgraphs is important in many applications. It is used, for example in the identification of network motifs (statistically overrepresented induced subgraphs of small size): a straightforward algorithm to find such motifs is to enumerate all connected induced subgraphs and to count how often each subgraph of order k occurs [127, 227]. A further application arises when semantic web data is searched using only keywords instead of structured queries [68]. In this application important triples are created and then these triples are incorporated into a graph. Furthermore, in this application, one is interested in all connected subgraphs not only induced ones. The enumeration of connected induced subgraphs can be used as a subroutine here, however, since all connected subgraphs can be obtained by enumerating all connected subgraphs of each connected induced subgraph. Finally, many fixed-cardinality optimization problems can be solved by an algorithm whose first step is to enumerate connected induced subgraphs of order k [150]. This algorithm can solve, for example, Connected Densest-k-Subgraph, the problem of finding a connected vertex set S of order k with a maximum number of edges having both endpoints in S. Experiments showed that enumeration-based algorithms can be competitive with Integer Linear Programs [151] underlining the importance of fast enumeration algorithms for connected induced subgraphs.

A closely related problem is the enumeration of all connected edge-induced subgraph. Recall that for an edge set F the *edge-induced subgraph* is the graph $G' := (V_F, F)$, where V_F is the set of endpoints of the edges in F. This problem is an impor-

tant subtask for determining frequent subgraphs in a given multilayer graph [202]. A *multilayer graph* G with r layers consists of one *layer* G_i for each $i \in [r]$. All layers have the same vertex set. A subgraph H is called *frequent* in G if in at least t layers H is a subgraph of G_i. The enumeration of frequent subgraphs in graphs with edge (and vertex) labels is an important task in bioinformatics [181, 182].

The fundamental graph problem of enumerating all connected induced subgraphs can be formalized as follows.

Exact Connected Induced Subgraph Enumeration (E-CISE)
Input: An undirected graph $G = (V, E)$ and an integer k.
Task: Enumerate all connected induced subgraphs of order k of G.

We call a connected subgraph of order k a *solution* in the following. At first sight, providing any non-trivial upper bounds on the running time of E-CISE seems hopeless: As evidenced by a clique on n vertices, graphs may have up to $\binom{n}{k}$ E-CISE solutions. Even very sparse graphs may have $\binom{n-1}{k-1}$ E-CISE solutions as evidenced by a star graph with $n-1$ leaves. It is maybe due to these lower bounds that, despite its importance, E-CISE has not received too much attention from the viewpoint of worst-case running time analysis.

One way to achieve relevant running time bounds is to consider degree-bounded graphs. Here, the number of solutions is much smaller than in general as shown by the following bound due to Bollobás [28].

Lemma 3.1 ([28, Equation 7])**.** *Let G be a graph with maximum degree Δ. Then the number of connected induced subgraphs of order k that contain some vertex v is at most $(e(\Delta-1))^{(k-1)}$. Hence, the overall number of connected induced subgraphs of order k in G is $\mathcal{O}((e(\Delta-1))^{(k-1)} \cdot (n/k))$ where n is the number of vertices of G.*

This observation can be exploited to obtain an algorithm for E-CISE that runs in $\mathcal{O}((e(\Delta-1))^{(k-1)} \cdot (\Delta+k) \cdot (n/k))$ time [150].

A second approach to provide non-trivial running time bounds is to prove upper bounds on the *delay* of the enumeration. A reverse search algorithm (see Section 2.4), however, achieves delay $\mathcal{O}(k \min(n-k, k\Delta)(k(\Delta + \log k) + \log n))$ [67].

Thus, k and Δ appear to be central parameters governing the complexity of E-CISE. Motivated by this observation, we aim to make further progress at exploiting small values of Δ and k.

Previous Work. Most known E-CISE algorithms follow the same strategy: starting from an initial vertex set $S \coloneqq \{v\}$ for some vertex v, build successively larger

connected induced subgraphs $G[S]$ until an order-k subgraph is found. Wernicke [227] describes a very simple procedure following this paradigm. The idea is to branch into the different possibilities to add one vertex u from $N(S)$. We refer to this procedure as `Simple`. Another popular enumeration algorithm is `Kavosh` [127] which also considers adding vertices of $N(S)$ but creates one branch for each subset of $N(S)$ that has size at most $k - |S|$.

A slightly different strategy is to first pick a vertex p of the current set S whose neighbors are added in the next step and then branch on the up to $(\Delta-1)$ possibilities for adding a neighbor of this vertex. The vertex p is called the *active* vertex of the enumeration. The corresponding algorithm, which we call `Pivot`, has worst-case running time $\mathcal{O}((4(\Delta-1))^k \cdot (\Delta+k) \cdot n)$ [149]. A variant of `Pivot` achieves the running time of $\mathcal{O}((e(\Delta-1))^{(k-1)} \cdot (\Delta+k) \cdot n/k)$ mentioned above [150]. This variant, which we call `Exgen`, generates *exhaustively* all subsets S' of $N(p) \setminus S$ of size at most $k-|S|$ and creates for each such set S' one branch in which S' is added to S.

Another variant is `BDDE` [167]. For a fixed vertex v, `BDDE` enumerates the connected subgraphs containing v for increasing subgraph orders. The main idea is to use two functions, one to discover new graph edges and one to copy already enumerated parts of the enumeration tree.

An output-sensitive algorithm for E-CISE with running time $\mathcal{O}(\sum_{G^* \in \mathcal{S}} |G^*|)$, where $\mathcal{S}$ is the set of all E-CISE solutions and $|G^*|$ is the total size of G^* was presented by Ferreira [79]. This overall running time is optimal when the task is to fully output all solutions, not only their vertex sets. The basic idea of this algorithm, which is closely related to `Simple`, is to create a binary search tree whose nodes represent connected sets S of G and whose leafs represent solutions. In each search tree node, the algorithm selects a vertex from v from $N(S)$ and branches into two cases: it first enumerates the solutions that contain $S \cup \{v\}$, and then those that contain S but not v. In addition, a certificate is used to ensure that in each node of the search tree, there exists at least one solution that contains S. Ferreira [79] does not bound the delay of this algorithm.

The known algorithms with polynomial delay [67] work differently. They use reverse search and the supergraph method [11]. There, for a given graph G and parameter k, the supergraph $\mathcal{G}$ contains a node for each E-CISE solution in G. Furthermore, two nodes in $\mathcal{G}$ are connected if and only if the corresponding connected subgraphs differ in exactly one vertex. The basic idea of reverse search is to explore the supergraph $\mathcal{G}$ efficiently. Let $|\mathcal{G}|$ denote the number of vertices in $\mathcal{G}$, that is, the number of E-CISE solutions. By using reverse search (see Section 2.4), one can enumerate all induced subgraphs of order *at most* k with polynomial delay [11]. When we are interested only in solutions of order *exactly* k, this algorithm is *not* output-

polynomial, that is, the running time is not bounded by a polynomial in the input and output size. Consequently, it does not achieve polynomial delay either. However, Elbassioni described two algorithms with polynomial delay for E-CISE [67]: The first variant, which we refer to as `RwD` (`Reverse Search with Dictionary`) has a delay of $\mathcal{O}(k \min(n-k, k\Delta)(k(\Delta + \log k) + \log n))$ and requires $\mathcal{O}(n + m + k|\mathcal{G}|)$ space where m is the number of edges in the input graph G. The second variant, which we refer to as `RwP` (`Reverse Search with Predecessor`), has a delay of $\mathcal{O}((k \min(n-k, k\Delta))^2(\Delta + \log k))$ and requires $\mathcal{O}(n+m)$ space [67]. Hence, the `RwD` algorithm has a better delay but requires exponential space, since $\mathcal{G}$ may grow exponentially with the size of G.

Another related important problem is the enumeration of all connected subgraphs of order *at most* k.

Bounded Connected Induced Subgraph Enumeration (B-CISE)

Input: An undirected graph $G = (V, E)$ and an integer k.
Task: Enumerate all vertex sets of connected induced subgraphs of order at most k of G.

Note that here we focus on enumerating the *vertex sets* instead of enumerating the *subgraphs* since the minimal delay for the subgraph enumeration problem is bounded by the maximal size of a solution which is $k \cdot \min(k, \Delta)$. For the vertex set enumeration problem, however, smaller delays are possible.

Avis and Fukuda described the first algorithm with polynomial delay for B-CISE. This algorithm is based on reverse search and has a delay of $\mathcal{O}(nm)$ [11]. Recently, the `RSSP` algorithm achieved a delay of $\mathcal{O}(n_c)$ [4], where n_c is the order of the largest connected component of G, for the special case when $k \geq n_c$, that is, when there is no size restriction. The basic idea of `RSSP` is also to use the reverse search framework but with a more strict neighborhood definition than Avis and Fukuda [11]. Furthermore, Haraguchi and Nagamochi [108] presented algorithms with polynomial delay and polynomial space for enumerating all vertex sets which induce an ℓ-edge-connected (ℓ-vertex-connected) subgraph. Observe that for $\ell = 1$ these two problems corresponds to B-CISE.

Finally, when the running time to output the solution is not counted, then all connected induced subgraphs can be enumerated in amortized time $\mathcal{O}(1)$ per connected induced subgraph [219].

Another, important problem is the enumeration of all connected edge-induced subgraphs of a graph. Salem et al. [202] provided an algorithm with delay $\mathcal{O}(m_c)$ to enumerate all edge-induced subgraphs of a graph G. Here, m_c is the largest number of edges of any connected component of G.

Our Results. We show how to adapt `Simple` and `Pivot` for E-CISE in such a way that the worst-case delay between the output of two solutions is $\mathcal{O}(k^2\Delta)$ and that the algorithms require $\mathcal{O}(n+m)$ space. This improves over the previous best delay bound of `RwD` [67] while requiring only linear space. As a side result, we show that these variants of `Simple` and `Pivot` achieve an overall running time of $\mathcal{O}((e(\Delta-1))^{(k-1)}\cdot(\Delta+k)\cdot n/k)$ and $\mathcal{O}((e(\Delta-1))^{k-1}\cdot\Delta\cdot n)$, respectively. For `Simple` this is the first running time bound, for `Pivot`, this is a substantial improvement over the previous bound. In addition, we further explore the connections between `Simple` and `Pivot` and show that a certain implementation of `Pivot` is in fact only a variant of `Simple`. Furthermore, we improve the delay of `RwP` to $\mathcal{O}(k^2 \min(n-k, k\Delta) \cdot \min(k\Delta, (n-k)(\Delta+\log k)))$ while still requiring only $\mathcal{O}(n+m)$ space.

W also give delay bounds for B-CISE. More precisely, we show that `Simple` and `Pivot` achieve a delay of $\mathcal{O}(k+\Delta)$. Since $k \leq n_c$ and $\Delta < n_c$ where n_c is the order of the largest connected component of the graph, we achieve the same delay of $\mathcal{O}(k+\Delta) = \mathcal{O}(n_c)$ if $k \in \mathcal{O}(n_c)$ or $\Delta \in \mathcal{O}(n_c)$, that is, in particular for $k = n_c$. If $k \ll n_c$ and $\Delta \ll n_c$ then we improve upon the best delay. Furthermore, observe that the best delay for B-CISE is much lower than for E-CISE.

Afterwards, we show that `Simple` and `Pivot` can also be used to enumerate all connected *edge-induced* subgraphs of size exactly (and also at most) k. We achieve the same delay for the edge-induced variants as we did for the vertex-induced variants. In particular, for the problem of enumerating all connected edge-induced subgraphs, our algorithms achieve a delay of $\mathcal{O}(\Delta+k)$ which improves upon the previous best delay of $\mathcal{O}(m_c)$ for this problem [202]. Here, m_c is the maximal number of edges of any connected component.

Finally, we show that `Simple` and `Pivot` can be adapted to enumerate all connected subgraphs with exactly (at most) k vertices. In these problems, we do not require the subgraphs to be induced.

We then evaluate the performance of different algorithms for E-CISE. For this, we implemented `Simple` and `Pivot` in Python and compare them experimentally with our own Python implementations of `Kavosh` [127], `Exgen` [150], `BDDE` [167], `RwD` [67], and `RwP` [67]. For $k \leq 10$, we observe that `RwD` and `RwP` are significantly slower than the other algorithms, which behave quite similarly in terms of overall running time. For very large k, that is, for k close to the order of the largest connected component n_c of G, `RwD` and `RwP` are again slower than the other algorithms and the differences between these algorithms are larger in this case. Here, `Pivot` has the best overall running times.

We also perform experiments for B-CISE. For these experiments we included `RSSP` in the comparison and excluded `RwD` and `RwP` since they were not designed for

this variant. The main result is that all algorithms behave roughly similar for this problem.

Overview. We first adapt `Simple` and `Pivot` to achieve polynomial delay for E-CISE. For this we provide in Section 3.1 a main algorithm loop which enumerates all connected induced subgraphs containing some vertex v and deletes v afterwards. Then, in Sections 3.2 (`Simple`) and 3.3 (`Pivot`) we provide obtain the claimed delay for E-CISE. Afterwards, in Section 3.4 we show that the delay of one of the algorithm of Elbassioni [67] can be improved slightly. In Section 3.5 we adapt `Simple` and `Pivot` to also achieve polynomial delay for B-CISE. Afterwards, in Section 3.6 we show that `Simple` and `Pivot` can also be used to enumerate connected edge-induced subgraphs. Then, in Section 3.7 we adapt these algorithms to the enumeration of connected)not necessarily induced) subgraphs. In Section 3.8 we provide experiments for the running times of several algorithms for E-CISE and B-CISE. Finally, in Section 3.9 we conclude this chapter with some open questions.

3.1 Main Principle of the Algorithm

Enumeration Trees and the Main Algorithm Loop. With the exception of `RwD` and `RwP`, the enumeration algorithms `Simple`, `Pivot`, `Exgen`, `Kavosh`, and `BDDE` use a search tree method which is called from a main loop whose pseudocode is given in Algorithm 3.1. Different algorithms, for example `Simple` or `Pivot`, can be used as `Enum-Algo` in Line 4 in Algorithm 3.1. For each vertex in the graph, Algorithm 3.1 creates a unique *enumeration tree*. In other words, Algorithm 3.1 produces $|V|-k+1$ enumeration trees. To avoid confusion, we refer to the vertices of the enumeration trees as *nodes*. Each node represents a connected subgraph $G[P]$ of order at most k. We refer to the vertex set S of this subgraph as *subgraph set* of the enumeration tree node. Roughly speaking, a node N is a child of another node M if the subgraph corresponding to M is an induced subgraph of the subgraph corresponding to N. The exact definition of child depends on the choice of `Enum-Algo`. A *leaf* is a node without any children. Furthermore, a leaf is *interesting* if P has size k; otherwise it is *boring*. A node *leads to an interesting leaf* if at least one of its descendants is an interesting leaf.

In the main algorithm loop, we enumerate for each vertex v of the input graph all E-CISE solutions containing v by calling the respective enumeration procedures; the first call of the enumeration procedure is the *root* of the enumeration tree and it represents the connected subgraph $G[v]$. After enumerating all solutions containing v,

Algorithm 3.1: The main loop for calling the enumeration algorithms; `Enum-Algo` can be any of `Simple`, `Pivot`, `Exgen`, `Kavosh`, and `BDDE`.

```
1 Algorithm Enumerate(G = (V, E))
2     while |V(G)| ≥ k do
3         choose vertex v from V(G)
4         enumerate all E-CISE solutions containing v with Enum-Algo
5         remove v from G
```

the vertex v is removed from the graph.

Cleaning the Graph. Observe that it is necessary to delete vertices of components with less than k vertices after one call of `Enumerate` to obtain a delay which is polynomial in $k + \Delta$: Otherwise, after some calls to `Enum-Algo` we may end up with a graph containing many vertices which are not contained in solutions (up to $\mathcal{O}(n)$). Starting the enumeration from these vertices one after the other would not give a delay that is polynomial in k and Δ. Hence, we show how to remove these connected components quickly. In the following we denote a connected component as *small* if it has at most $k - 1$ vertices.

Lemma 3.2. *Let G be a graph such that each connected component has order at least k and let v be an arbitrary vertex of G. In $\mathcal{O}(k^2\Delta)$ time we can delete every vertex of $G - v$ that is in a small connected component.*

Proof. The only vertices of $G - v$ that are in small connected components, are those that are in the same connected component as v in G. We may thus check for each connected component of $G - v$ which contains at least one neighbor of v whether this connected component is small. Using depth-first search, this check needs $\mathcal{O}(k^2)$ time per neighbor. Moreover, for each neighbor which is in a small connected component, we can remove the connected component from G in $\mathcal{O}(k^2)$ time. Since v has at most Δ neighbors in G, the total running time of the algorithm is $\mathcal{O}(k^2\Delta)$. □

3.2 Polynomial Delay with Simple

We now adapt the `Simple` algorithm of Wernicke [227] to obtain a polynomial delay algorithm; the pseudocode is shown in Algorithm 3.2. In `Simple`, we start with a single vertex v and find successively larger connected subgraphs containing v. The subgraph set is denoted by P. Furthermore, the set $X \subseteq N(P)$, called *extension set*,

Algorithm 3.2: The `Simple` algorithm; the initial call is `Simple`$(\{v\}, N(v))$.

```
1  Algorithm Simple(P, X)
2      if |P| = k then
3          output P
4          return True
5      hasIntLeaf := False
6      while X ≠ ∅ do
7          u := choose last vertex from X
8          delete u from X            ▷ The current set P will be extended
9          X' := X ∪ (N(u) \ N[P])
10         if Simple(P ∪ {u}, X') = True then
11             hasIntLeaf := True
12         else
13             return hasIntLeaf
14                                    ▷ Stop recursion if no new solution found
15     return hasIntLeaf
```

contains those neighbors of P which can be added to P to enlarge the subgraph $G[P]$. In Lines 7-9, when putting $u \in X$ in the set P, we remove u from X and add to X each neighbor of u which is not in $N[P]$. Removing u from X implies that u cannot be added to the subgraph set again within the subtree rooted in this node.

Next, we describe our change to this algorithm. Lines 4, 5, and 10–15 of Algorithm 3.2 and the use of the boolean flag variable hasIntLeaf are not part of the plain version of `Simple` [227]. In these lines, a new pruning rule is performed; this rule is necessary to establish polynomial delay for E-CISE. In the following, we describe the idea of this rule.

Consider a path $T_1, \ldots, T_i$ from the root T_1 to a node T_i of the enumeration tree. We denote the subgraph set of a node T_i by P_i and its extension set by X_i. To avoid unnecessary recursions, we check after each recursive call of `Simple` in node T_i whether this call reported a new solution. If not, we return in T_i to its parent T_{i-1}. First, we prove that this pruning rule is correct. Recall that a leaf T_j is called interesting if the corresponding subgraph set P_j is a solution for E-CISE(that is, if $|P_j| = k$) and that T_j is called boring otherwise.

Lemma 3.3. *Let T_i be a node in the enumeration tree of `Simple`. If the output of a recursive call of `Simple` in node T_i is empty, then no subsequent recursive call of*

`Simple` in node T_i leads to an interesting leaf.

Proof. Let $T_i, \ldots, T_j$ denote the path in the enumeration tree from node T_i to a leaf T_j, where node $T_{\ell+1}$ is the first child of node T_ℓ for each $\ell \in \{i+1, \ldots, j-1\}$. By assumption, the leaf T_j is boring, that is, $|P_j| < k$. This implies that T_j has an empty extension set X_j. Hence, the number $j-i$ of vertices that are added between node T_i and leaf T_j is equal to the number of those vertices in the graph $G - (N[P_i] \setminus X_i)$ which are in the same connected component as P_i. In other words, adding all possible vertices in node T_i does not give a connected subgraph of order at least k.

Observe that vertex u that was added to the subgraph set P_{i+1} of node T_{i+1} is removed from X_i after the creation of T_{i+1} and cannot be part of the subgraph set in any node rooted in T_i again. Hence, the number of vertices in $G-(N[P_i] \setminus X_i)$ which are in the same connected component as P_i is at least one more than the number of vertices in $G - (N[P_i] \setminus \{X_i \setminus \{u\}\})$ which are in the same connected component as P_i. Thus, also the next child of node T_i and any further child of T_i will not lead to an interesting leaf. □

By the above, we can abort the enumeration in node T_i and return to the parent T_{i-1} of T_i as soon as one of the recursive branches fails to output a solution.

To check efficiently whether the pruning rule applies, we do the following: Each enumeration tree node T has a boolean variable hasIntLeaf initialized with `False` in Line 5. As soon as at least one recursive call has an interesting leaf, the variable hasIntLeaf is set to `True`. Therefore, the algorithm correctly returns whether or not T leads to an interesting leaf. If some recursive call does not lead to an interesting leaf, then the algorithm returns immediately to the parent of node T which is correct due to Lemma 3.3. The overall overhead for performing the pruning is only a constant factor.

With this pruning rule at hand we now show that `Simple` achieves a polynomial delay. To achieve the claimed delay of $\mathcal{O}(k^2\Delta)$, we present a new data structure to store the extension set during the algorithm. In the following, we denote by p_i the vertex which was added to the subgraph set P_i when T_i is created. In other words, if T_{i-1} is the parent of T_i, then $P_i \setminus P_{i-1} = \{p_i\}$. First, we prove that for a node T_i in the enumeration tree we need $\mathcal{O}(\Delta)$ time to either compute the sets P_{i+1} and X_{i+1} of its next child T_{i+1} or to restore the sets P_{i-1} and X_{i-1} of its parent node T_{i-1}.

Lemma 3.4. *`Simple` can be implemented in such a way that for every node T_i of the enumeration tree, we need $\mathcal{O}(\Delta)$ time to either compute the next child T_{i+1} or to restore the parent T_{i-1} and that the overall space needed is $\mathcal{O}(n+m)$.*

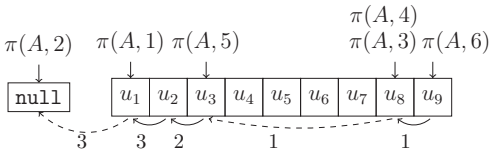

Figure 3.1: An example for the pointer movement: Pointer $\pi(A, 6)$ points to u_9, an exclusive neighbor of p_6. Before adding u_9 to the subgraph set P_6, we move pointer $\pi(A, 6)$ to the left to u_8, an exclusive neighbor of vertex p_3. Since T_5 is the parent of T_6 we move $\pi(A, 6)$ to u_3 which is the position of pointer $\pi(A, 5)$. Next, we create a child of T_6 by adding u_9 to the subgraph set P_6. The next time we are in node T_6, we move $\pi(A, 6)$ one step to the left to vertex u_2 and create a child of T_6 by adding u_3 to P_6. After returning from this child, we move $\pi(A, 6)$ to vertex u_1 which is an exclusive neighbor of vertex p_1. Since T_2 is the parent of T_3 we move $\pi(A, 6)$ to the position of $\pi(A, 2)$. Afterwards, we create a child by adding u_2 to P_6. The next time we come back to node T_6, we delete pointer $\pi(A, 6)$, since $\pi(A, 6)$ points to `null`, and return to the parent T_5 of node T_6.

Proof. We describe the data structures that we use to fulfill the running time and space bounds of the lemma. To check whether a vertex is in some extension set, we color some vertices of G with $k+1$ colors $c_0, \ldots, c_k$ as follows. Following the notation of Wernicke [227], for a node T_i, we call the vertices which are in $N[P_i] \setminus N[P_{i-1}]$ where T_{i-1} is the parent of T_i the *exclusive neighbors of* p_i. These are exactly the vertices that are added to X_{i-1} in Line 9 of Algorithm 3.2 to construct the extension set X_i for the node T_i. Throughout the algorithm we maintain the following invariant:

> Let $T_1, \ldots, T_i$ for $i \leq k$ be the path from the root T_1 to a node T_i. The vertex p_1 has color c_0. A vertex v has color c_i, $i \geq 1$, if and only if v is an exclusive neighbor of p_i.

Altogether, for $0 \leq j \leq k$, the colors $c_0, \ldots, c_j$ represent the vertices in $N[P_j]$. It is necessary to use $k+1$ different colors to determine in which node a vertex was added to the extension set. Note that every vertex may have at most one color.

The extension sets of all nodes on the path from the root T_1 to an enumeration tree node T_i are represented by an array A of length $k\Delta$ with up to k pointers pointing to positions of A. There is one pointer $\pi(A, i)$ corresponding to T_i and one pointer $\pi(A, j)$ for each ancestor T_j of T_i. An entry of A is either empty or contains a pointer to a vertex of the extension set X_i. In Line 9 of Algorithm 3.2 when the new extension set X' is created from X, the new vertices of $N(u) \setminus N[P]$ replace the

left-most empty entries of A. Pointer $\pi(A, i)$ points to the vertex x in the extension set X_i which will be added to P_i in the *next* recursive call of `Simple` in node T_i. If at node T_i already all children of T_i have been created, then $\pi(A, i)$ points to `null`. Hence, we may check in constant time whether T_i has further children and restore the sets P_{i-1} and X_{i-1} to the parent T_{i-1} if this is not the case.

In addition to A, we use two further simple data structures: The subgraph set P_i at a node T_i is implemented via a stack Q that is modified in the course of the algorithm with the top element of the stack being p_i. Also, for each node T_i, we create a list L_i of the exclusive neighbors of p_i. This list is necessary to undo some later operations. We now describe how these data structures are maintained throughout the traversal of the enumeration tree.

Initialization. At the root T_1 of the enumeration tree, we initialize A as follows: add all neighbors of the start vertex $p_1 \coloneqq v$ to A, set pointer $\pi(A, 1)$ to the last non-empty position in A. These are precisely the vertices of the exclusive neighborhood of v and these vertices occupy the first $\pi(A, 1)$ positions of A. The stack Q consists of the vertex v and L_1 contains all neighbors of v.

Creation of new children. As discussed above, a node T_i has a further child T_{i+1} if $\pi(A, i)$ points to an index containing some vertex x. We create child T_{i+1} as follows:

1. move the pointer $\pi(A, i)$ to the left,
2. check whether x is an exclusive neighbor of p_i, and remove x from A if this is the case, and
3. create the child T_{i+1} with $p_{i+1} = x$ and enter the recursive call for T_{i+1}.

We now specify how to *move the pointer $\pi(A, i)$ to the left* when it currently points to vertex x of color c_ℓ. An example of the pointer movement is given in Figure 3.1. That is, x is an exclusive neighbor of p_ℓ for some $\ell \leq i$. Note that if x is an exclusive neighbor of p_i, we have $i = \ell$. If x is contained in the first entry of A, then redirect $\pi(A, i)$ to `null`. Otherwise, decrease the position of $\pi(A, i)$ by one. If $\pi(A, i)$ now points to a position containing a vertex y of color c_j such that $j < \ell$, then move $\pi(A, i)$ to the position that $\pi(A, \ell - 1)$ points to. Since y is an exclusive neighbor of p_j pointer $\pi(A, j)$ points to y. Observe that if $j = \ell - 1$ this means that the pointer does not move in the second step.

We now describe how the algorithm creates a child T_{i+1} of T_i after fixing $p_{i+1} \coloneqq x$ as described above. If node T_{i+1} is an interesting leaf, that is, if $i = k - 1$, we output $P_{i+1} \cup \{x\}$ and return to node T_i. Otherwise, we add vertex x to the stack Q representing the subgraph set and create an initially empty list L_{i+1}. Then we update A so that it represents X_{i+1}. For each neighbor u of x, check if u has some

color c_j. If this is not the case, then assign the color c_{i+1} to u and add u to L_{i+1}. Now, store the vertices of L_{i+1} in the left-most non-empty entries of A. Finally, create the pointer $\pi(A, i+1)$ and let it point to the last non-empty position in A. Observe that this procedure runs in $\mathcal{O}(\Delta)$ time.

Restoring the parent. Finally, we describe how the algorithm returns to the parent T_{i-1} of a node T_i. Note that the case that T_i is an interesting leaf was already handled above. In the following, assume that T_i is not an interesting leaf. When returning to T_{i-1}, first delete the last element p_i of stack Q. Then, for each vertex in L_i, we remove its color c_i. Finally, remove pointer $\pi(A, i)$ from array A. Observe that this can be done in $\mathcal{O}(\Delta)$ time as well.

We conclude that the overall running time is $\mathcal{O}(\Delta)$ as claimed. Moreover, the size of stack Q is bounded by k, array A has a length of $\min(k\Delta, n)$, and the sum of the sizes of all lists L_i is at most $\min(k\Delta, n)$. Hence, `Simple` needs $\mathcal{O}(n+m)$ space.

We now prove that the pointer structure faithfully represents the extension sets during the course of the algorithm. More precisely, we show that each pointer $\pi(A, i)$ visits all the vertices that are contained in the extension set X_i when T_i is created. To this end, define inductively for all $\pi(A, j)$ pointing on the array A, the following subsets $A_{\leq j}$ of entries of A inductively. The set $A_{\leq 1}$ contains all entries to the left of $\pi(A, 1)$ including the entry that $\pi(A, 1)$ points to. The set $A_{\leq j}$ contains all entries of A that are to the left of $\pi(A, j)$ and either in $A_{\leq j-1}$ or exclusive neighbors of T_j. We now claim the following invariant during the algorithm by induction over the operations of the algorithm.

> Let T_i be a enumeration tree node, and let $T_1, \ldots, T_i$ denote the nodes on the path from the root T_1 of the enumeration tree to node T_i. At any point in time $A_{\leq i}$ contains exactly the set X_i.

The claim is obviously true for the root T_1: Initially, A contains exactly the vertices of $N(p_1)$ and all of them are to the left of the position of $\pi(A, 1)$. Every time the pointer $\pi(A, 1)$ moves to the left, it moves by exactly one position and removes exactly one vertex from X_1 as prescribed in Line 8 of Algorithm 3.2.

Now, assume by induction that the claim holds for all T_j with $j < i$. When creating T_i, the extension set X_i may, according to the pseudocode of `Simple`, contain all vertices which are in X_{i-1} or exclusive neighbors of p_i. When creating T_i, the pointer $\pi(A, i)$ points to the rightmost non-empty position in the array A. Thus $A_{\leq i}$ contains all vertices which are exclusive neighbors of p_i or are contained in $A_{\leq i-1}$. By the inductive hypothesis, the latter set contains exactly the vertices of X_{i-1}. When the pointer $\pi(A, i)$ is moved to the left, this corresponds to removing the element that $\pi(A, i)$ points to from X_i as prescribed in Line 8 of Algorithm 3.2 (where we

remove u). To prove the claim that the procedure of moving to the left is correct, we show that the pointer stops at the rightmost position of A containing an element of X_i. We distinguish three cases:

Case 1: $\pi(A, i)$ now points to some vertex with color c_i. Then this vertex is an exclusive neighbor of T_i and the pointer has moved exactly one position to the left. Hence, the pointer stops at the rightmost position of A containing an element of X_i.

Case 2: $\pi(A, i)$ now points for the first time to some vertex x with color c_j such that $j < i$. Then, $\pi(A, j)$ points to the same position as $\pi(A, i)$, since x is the rightmost remaining vertex of the extension set X_j. If $j = i - 1$, then the pointer position is not changed further and $\pi(A, i)$ now points to the rightmost exclusive neighbor of T_{i-1} which is the rightmost vertex of X_i in A since A contains no exclusive neighbors of T_i anymore. Otherwise, the pointer jumps to the position of $\pi(A, i-1)$. Since all elements of X_{i-1} are contained in $A_{\leq i-1}$ and since, by induction, $\pi(A, i-1)$ points to the rightmost element of X_{i-1} we have that $\pi(A, i)$ points at the rightmost position of A containing an element of X_i.

Case 3: $\pi(A, i)$ was moved from a vertex with color c_j such that $j < i$ to the left. In that case, the movement of $\pi(A, i)$ is exactly the same as the movement of $\pi(A, i - 1)$ when the algorithm visits node T_{i-1}. By induction, we assume that this movement visits all the vertices of X_{i-1}.

Hence, when moving the pointer to the left, we do not miss an element of X_i and we only stop at elements of X_i. □

With this running time bound to compute the next child or to restore the parent at hand, we may now prove the claimed delay. Recall that for this it is essential to remove small connected components, that is, those with size at most $k - 1$.

Theorem 3.5. *`Enumerate` with `Simple` solves* E-CISE *for any graph G where each connected component has order at least k and the maximum degree is Δ with delay $\mathcal{O}(k^2\Delta)$ and space $\mathcal{O}(n + m)$.*

Proof. `Enumerate` chooses an arbitrary start vertex v. According to Lemma 3.2, after the deletion of vertex v, we can delete every vertex of each connected component with less than k vertices in $\mathcal{O}(k^2\Delta)$ time. Thus, it is sufficient to bound the time which is needed to output the next solution within `Simple`.

We compute the sets P_1 and X_1 in $\mathcal{O}(\Delta)$ time. Since the algorithm of Lemma 3.2 was applied, by adding $k - 1$ vertices, we obtain a solution for E-CISE. The corresponding connected induced subgraph can be output in $\mathcal{O}(k)$ time. Hence, the first solution for a call of `Enumerate` will be output in $\mathcal{O}(k\Delta)$ time.

Next, we show that in $\mathcal{O}(k^2\Delta)$ time we either find a new solution for E-CISE or end this call of `Enumerate`. For this we show that in $\mathcal{O}((k - j)k\Delta)$ time we can

restore the sets P_j and X_j of node T_j for some $j < k$ or output the next solution. Showing this statement is sufficient since the depth of the enumeration tree is at most k and a return in the root means that no further induced subgraph of size k containing p_1 exists.

We show this statement by induction on j. Let T_k be an interesting leaf. According to Lemma 3.4, the sets P_{k-1} and X_{k-1} of node T_{k-1} can be restored in $\mathcal{O}(\Delta)$ time. Assume the sets P_j and X_j are restored in $\mathcal{O}((k-j)k\Delta)$ time. Every time we call `Simple` recursively, we add exactly one vertex to the subgraph set. Hence, we need at most k iterations to reach a leaf T_ℓ. If T_ℓ is interesting, that is if $\ell = k$, then we output the corresponding connected induced subgraph in $\mathcal{O}(k \min(k, \Delta))$ time. Thus, in this case the algorithm has a delay of $\mathcal{O}((k-j+1)k\Delta)$ to output the next solution. If T_ℓ is boring, that is if $\ell < k$, then according to Lemma 3.3 the pruning rule applies to each node T_q on the path from T_ℓ to T_j since no other subsequent child of node T_q yields a path to an interesting leaf. Hence, we will return in altogether $\mathcal{O}(k\Delta)$ time to the parent T_{j-1} of node T_j. Hence, the sets P_{j-1} and X_{j-1} can be restored in $\mathcal{O}((k-j+1)k\Delta)$ time. Thus, in $\mathcal{O}(k^2\Delta)$ time we either output the next solution or return in the root T_1 and end this call of `Enumerate`. Hence, the overall delay is $\mathcal{O}(k^2\Delta)$. The space complexity follows from Lemma 3.3. □

We can use Lemma 3.4 also to bound the overall running time of the algorithm.

Proposition 3.6. *`Enumerate` with `Simple` has running time* $\mathcal{O}((e(\Delta-1))^{k-1} \cdot (\Delta + k) \cdot n/k)$.

Proof. Each connected induced subgraph with at most k vertices is output exactly once [227]. Hence, for two different nodes T and Q in the enumeration tree, we have $P_T \neq P_Q$. In other words, each enumeration tree node corresponds to a different connected subgraph of order at most k. According to Lemma 3.1, the overall number of these subgraphs containing some vertex v is $\mathcal{O}(\sum_{i=1}^{k}(e(\Delta-1))^{i-1}) = \mathcal{O}((e(\Delta-1))^{k-1})$ where the equality follows from the fact that $2(e(\Delta-1))^{i-1} < (e(\Delta-1))^i$ for $\Delta \geq 2$. Consequently, the number of search tree nodes in the enumeration trees in all calls of `Enumerate` with `Simple` is $\mathcal{O}((e(\Delta-1))^{k-1} \cdot n/k)$.

Now, we bound the time per node T in the enumeration tree. Let p_T be the vertex that was added to the subgraph set to create the node T. Determining the neighbors of vertex p_T and adding the exclusive neighbors of p_T (that are those without a color) to X_T needs $\mathcal{O}(\Delta)$ time. For each vertex z in X_T we make a recursive call where we add z to the subgraph set. The recursive call includes updating the subgraph set and the extension set and can be done in $\mathcal{O}(\Delta)$ time per call. By charging this running time to the corresponding child in the enumeration tree, we obtain a running

Algorithm 3.3: The `Pivot` algorithm; the initial call is `Pivot`$(\{v\}, \{\}, v, \{\})$.

```
1  Algorithm Pivot(P, S, p, F)
2      if |P ∪ S| = k then
3          output P ∪ S
4          return
5      if p = null then
6          if P ≠ ∅ then
7              p ≔ choose some element of P
8          else
9              return
10     for z ∈ N(p) \ {P ∪ S ∪ F} do
11         Pivot(P ∪ {z}, S, p, F)
12         F ≔ F ∪ {z}
13     Pivot(P \ {p}, S ∪ {p}, null, F)
14     return
```

time of $\mathcal{O}(\Delta)$ per enumeration tree node. To output a solution needs $\mathcal{O}(k)$ time we obtain a running time of $\mathcal{O}(k + \Delta)$ per enumeration node. The overall running time follows. □

3.3 From Pivot to a Variant of Simple

We now adapt `Pivot` of Komusiewicz and Sorge [149] to obtain polynomial delay and a better running time bound. More precisely, we show that a careful implementation of an adaption of `Pivot` actually turns it into a variant of `Simple`. The pseudo code of `Pivot` as described by Komusiewicz and Sorge [149] can be found in Algorithm 3.3; the algorithm works as follows. In each enumeration tree node, the subgraph set is partitioned into two sets P and S. The set P contains those vertices whose neighbors may still be added to extend the subgraph set and set S contains the other vertices of this subgraph, that is, no neighbor of S may be added to the subgraph. Moreover, we have a set F containing further vertices that may not be added to the connected subgraph. Each node in the enumeration tree has an *active* vertex from the set P whose neighbors will be added to the subgraph. After considering each possible neighbor, the vertex becomes *inactive* and is removed from P and added to S. This version of the algorithm has a running time bound of $\mathcal{O}(4^k(\Delta - 1)^k n(n + m))$ [149]

Algorithm 3.4: An adaptation of `Pivot` without active vertex; the initial call is `Pivot`$(\{v\}, \emptyset, \emptyset)$.

```
1  Algorithm Pivot(P, S, F)
2      if |P ∪ S| = k then
3          output P ∪ S
4          return
5      while P ≠ ∅ do
6          p := choose first element of P
7          foreach z ∈ N(p) \ (P ∪ S ∪ F) do
8              Pivot(P ∪ {z}, S, F)
9              F := F ∪ {z}
10         P := P \ {p}
11         S := S ∪ {p}
12     return
```

and no polynomial delay. Next, we present our adaption of `Pivot`.

The pseudocode of our adaption of `Pivot` can be found in Algorithm 3.4. As we show, this variant already has, up to polynomial factors, a optimal overall running time. The adaption, however, does not yet achieve polynomial delay. Since our implementation of this algorithm eventually leads to a variant of `Simple`, we omit the proof of the overall running time and show running time bounds only for the final version. Nevertheless, we believe it is instructive to discuss this intermediate version of the algorithm. Consider a path $T_1, \ldots, T_i$ from the root T_1 to a node T_i of the enumeration tree. We will not associate enumeration tree nodes with active vertices. Instead, with each node T_i we associate the set P_i which is the subset of the subgraph set which can have further neighbors, the set S_i which is the remaining subgraph set, and the set F_i which is the set of forbidden vertices. Now instead of creating a new child when choosing a new active vertex we are using the while-loop starting in Line 5. In this while-loop we do the following until P_i is empty: Pick the vertex $p \in P_i$ that was added first to P_i. That is, if p_i is the ith vertex that was added to the subgraph set, then p_i becomes the active vertex p exactly after p_{i-1} moves from P to S. Otherwise, the algorithm is the same as the original one. Observe that when creating a child in Line 8, the active vertex (which is now only implicitly given) remains the same by the choice of p in Line 6.

We now further modify this variant of `Pivot`. Assume that the current vertex p is the ith vertex p_i that was added to P. The main idea is to show that the set $N(p) \setminus (P \cup S \cup F)$ in Line 7 from which the next vertex z is chosen can be computed

already when p_i is added to P. By the convention that p_i is chosen as the first vertex of P, the set S when p_i becomes active is predetermined, it is exactly $\{p_1, \ldots, p_{i-1}\}$. Moreover, whenever a vertex p_j, $j < i$, moves from P to S in Line 10, every neighbor of p_j is either in $P \cup S$ or in F. Hence, the set $N(p) \setminus (P \cup S \cup F)$ for $p = p_i$ is exactly the set $N(p_i) \setminus N[P \cup S]$. Thus, instead of saving the set F of forbidden vertices, we may work with a representation of an extension set X as in `Simple`. When adding a vertex z to the subgraph set, we add exactly those neighbors of z to the extension set X that are not neighbors of any vertex in $P \cup S$, that is, we add the vertices in $N(z) \setminus N[P \cup S]$.

As for `Simple`, the extension set X will be represented by an array A with several pointers pointing on this array. The difference is that these pointers move through A in forward direction, that is, from low indices to high indices. The subgraph set will be denoted by P only, that is, there is no need to store the set S anymore; it is implicitly represented by the position of a pointer on A. The same is true for the set F which is also not stored explicitly anymore. The pseudocode of this new implementation of `Pivot` is shown in Algorithm 3.5, in light of the above discussion, we will call it `Simple-Forward`. To highlight the differences between `Simple` and `Simple-Forward`, we emphasize that in `Simple-Forward` X has an order by using list notation, that is, initially X is a list of the neighbors of $v = p_1$, new vertices in X are appended and the next vertex is chosen from the front of the list instead of from the back as in `Simple`. Before proving the running time bounds for `Simple-Forward`, let us remark that an inspection showed that the implementation of `Simple` in the FANMOD tool of Wernicke and Rasche [228] is essentially the same as `Simple-Forward` except for the parts that are relevant for the pruning rules which are needed to establish polynomial delay (Lines 4, 5, and 10–15).

Next, we prove that with suitable data structures for maintaining the sets P and X during the enumeration, we can quickly traverse the enumeration tree.

Lemma 3.7. *`Simple-Forward` can be implemented in such a way that for every node T_i of the enumeration tree, we need $\mathcal{O}(\Delta)$ time to either compute the next child T_{i+1} or to restore the parent T_{i-1} and that the overall space needed is $\mathcal{O}(n+m)$.*

Proof. We use the same data structures as described in Lemma 3.4: an array A for the representation of X, for each i a list L_i representing the exclusive neighborhood of a vertex p_i, a stack Q representing the subgraph set P, and a coloring of the vertices to allow for an $\mathcal{O}(1)$-time test for containment in the exclusive neighborhood of some P_i. There is one difference, however: Instead of using $k+1$ colors, one for each exclusive neighborhood, we use only one color c for all vertices in $N[P_i]$. Next, we describe how these data structures are maintained during the enumeration tree.

Algorithm 3.5: The `Simple-Forward` algorithm, an implementation of `Pivot`; the initial call is `Simple-Forward`$(\{v\}, N(v))$.

```
1  Algorithm Simple-Forward(P, X)
2      if |P| = k then
3          output P
4          return True
5      hasIntLeaf ≔ False
6      while X ≠ ∅ do
7          u ≔ choose first vertex from X
8          remove u from X            ▷ The current set P will be extended
9          X' ≔ X with N(u) \ N[P] appended
10         if Simple-Forward(P ∪ {u}, X') = True then
11             hasIntLeaf ≔ True
12         else
13             return hasIntLeaf
14                                    ▷ Stop recursion if no new solution found
15     return hasIntLeaf
```

Initialization. At the root T_1 of the enumeration tree, we initialize A as follows: add all neighbors of the start vertex $p_1 \coloneqq v$ to A, set pointer $\pi(A, 1)$ to $A[1]$. These are precisely the vertices of the exclusive neighborhood of v. The stack Q consists of the vertex v and L_1 contains all neighbors of v.

Creation of new children. A node T_i has a further child T_{i+1} if $\pi(A, i)$ points to an index of A containing some vertex x. We create child T_{i+1} as follows: If x is the last entry of A, redirect $\pi(A, i)$ to `null`. Otherwise, move $\pi(A, i)$ one position to the right. Afterwards, create the child T_{i+1} with $p_{i+1} \coloneqq x$ as follows. If node T_{i+1} is an interesting leaf, that is, if $i = k - 1$, we output $P_{i+1} \cup \{x\}$ and return to node T_i. Otherwise, we add vertex x to the stack Q representing the subgraph set and create an initially empty list L_{i+1}. Then we update A so that it represents X_{i+1}: For each neighbor u of x, check if u has color c. If this is not the case, then assign u with color c and add u to L_{i+1}. Now store the vertices of L_{i+1} in the left-most non-empty entries of A. Finally, create the pointer $\pi(A, i + 1)$ and let it point to the same position as pointer $\pi(A, i)$, then enter the recursive call for T_{i+1}. Observe that this procedure runs in $\mathcal{O}(\Delta)$ time.

Restoring the parent. We describe how the algorithm returns to the parent T_{i-1} of a node T_i; the case that T_i is an interesting leaf was already handled above. Hence, assume that T_i is not an interesting leaf. When returning to T_{i-1}, first delete the

last element of stack Q. Then, for each vertex in L_i, we remove its color c. This procedure runs in $\mathcal{O}(\Delta)$ time.

The overall space complexity of $\mathcal{O}(n+m)$ follows by the same arguments as in the proof of Lemma 3.4.

It remains to show that the pointer structure faithfully represents the extension sets during the course of the algorithm. We have to show that each pointer $\pi(A,i)$ visits all vertices contained in the extension set X_i when node T_i is created. By $A_{>i}$ we denote the set of vertices in A beginning at $\pi(A,i)$ and ending at $A[|L_1|+\ldots+|L_i|]$. Note that $A[1,\ldots,|L_1|+\ldots+|L_i|]$ represents $N[P_i] \setminus \{p_1\}$. Similar to `Simple` we show the following invariant during the algorithm by induction over the operations of the algorithm.

> Let T_i be a enumeration tree node and let $T_1,\ldots,T_i$ denote the nodes on the path from the root T_1 of the enumeration tree to T_i. At any point in time $A_{>i}$ contains exactly the set X_i.

Similar to `Simple` the statement is obviously true for the root T_1 with the difference that $\pi(A,1)$ points to $A[1]$. Now assume the claim holds for all T_j with $j<i$. When node T_i is created, the corresponding extension set is $X_i \coloneqq X_{i-1} \cup N(P_i) \setminus N[P_{i-1}]$. According to the induction hypothesis, $A_{>i-1}$ correctly represents X_{i-1}. Furthermore, $L_i = N[P_i] \setminus N[P_{i-1}]$. Hence, when T_i is created, $A_{>i}$ correctly represents X_i. Each time a child T_{i+1} of node T_i is created, pointer $\pi(A,i)$ is moved exactly one position to the right. Hence, pointer $\pi(A,i)$ correctly represents X_i and thus visits all vertices in X_i. □

The correctness of the pruning rule for `Simple-Forward` can be proven analogously as the correctness of the pruning rule for `Simple` in Lemma 3.3 . Moreover, the proof of the delay bound of `Simple` in Theorem 3.5 also applies to `Simple-Forward`. Hence, we obtain directly obtain the following bound.

Corollary 3.8. *`Enumerate` with `Simple-Forward` solves* E-CISE *for any graph G where each connected component has order at least k and the maximum degree is Δ with delay $\mathcal{O}(k^2\Delta)$ and space $\mathcal{O}(n+m)$.*

Finally, because of Lemma 3.7, the proof of Proposition 3.6, which bounds the overall running time for `Simple`, applies also to `Simple-Forward`.

Corollary 3.9. *`Enumerate` with `Simple-Forward` has running time $\mathcal{O}((e(\Delta-1))^{k-1} \cdot (\Delta+k)\cdot n/k)$.*

3.4 Enumeration via Reverse Search

In this section, we describe the reverse search algorithms of [67]. Moreover, we present a small modification of one of the algorithms that leads to an improved delay bound for the case of small k.

3.4.1 Reverse Search with Dictionary (RwD)

The reverse search method which is also referred to as the supergraph method enumerates all solutions by traversing the supergraph $\mathcal{G}$ where every solution corresponds to exactly one node of $\mathcal{G}$. The pseudocode of the first of the two algorithms, `RwD`, is shown in Algorithm 3.6.

During the algorithm, each node in $\mathcal{G}$ gets the following labels: *visited* means that the node was visited by the algorithm, *discovered* means that the node was not yet visited but is a neighbor of an already visited node, and a node is *not discovered* otherwise. The algorithm saves all visited and discovered nodes in a set $\mathcal{K}$. Furthermore, all discovered nodes are stored in a queue Q .

In the first step, we have to determine an initial connected order-k subgraph in G, so we determine a start node T in $\mathcal{G}$. This can be done with depth-first search. Afterwards, only node T is assigned with label discovered and all remaining nodes have the label not discovered. So, initially the queue Q and the set $\mathcal{K}$ consist of node T.

As long as the queue Q is not empty we do the following: We remove the first node T of Q. Let S denote the solution represented by T. We output S and then we have to determine all neighbors of T in the supergraph $\mathcal{G}$ and add them to Q if they are not already discovered or visited. To this end, we subsequently remove each vertex v from S to construct connected induced subgraphs of size k containing $S' \coloneqq S \setminus \{v\}$. Observe that the induced subgraph $G[S']$ may be disconnected. Hence, we first determine the connected components of $G[S']$. Afterwards, we determine the set N containing all vertices of $V \setminus S$ that have at least one neighbor in each connected component of $G[S']$. We call the set N the *common neighborhood* of the connected components of $G[S']$. In the algorithm of Elbassioni [67], the common neighborhood is determined as follows: Check each vertex u of $V \setminus S$ and determine whether it has at least one neighbor in each connected component of S'. Each vertex u in N extends $G[S']$ to another solution. To output each solution exactly once, we use the set $\mathcal{K}$, that is, we check whether a solution has already been discovered or visited. A node corresponding to $S' \cup \{u\}$ is added to the queue only if it is not discovered.

Algorithm 3.6: The `RwD` algorithm.

```
1  Algorithm RwD(G, k)
2      Queue Q := ∅, K := ∅                ▷ K saves the enumerated solutions
3      foreach connected component C in G do
4          S := lexicographically largest solution in C
5          Q.append(S), K := K ∪ {S}
6          while Q ≠ ∅ do
7              S := Q.get()
8              output S
9              for vertex v ∈ S do
10                 S' := S \ {v}
11                 N := common neighborhood of connected components of S'
12                 for vertex w ∈ N do
13                     S'' := S' ∪ {w}
14                     if S'' ∉ K then
15                         Q.append(S''), K = K ∪ {S''}
```

3.4.2 Reverse Search with Predecessor (RwP)

This algorithm is almost the same as algorithm `RwD`. The main difference is the following: Instead of using the set $\mathcal{K}$ to save all visited and discovered nodes (which requires exponential space) it uses a *predecessor function* for solutions. The basic idea of this method in this context is the following: All solutions are sorted lexicographically and each solution has a unique *predecessor*. Assume that we are currently adding the neighbors of a node A in $\mathcal{G}$. When we find a new candidate B for a connected order-k subgraph, we only put B into the queue of discovered nodes if the predecessor of B is A.

Now we explain this method in more detail: We apply DFS search on the graph G and every vertex in G is assigned an index when it is discovered by this DFS. More precisely, the first vertex is assigned with the highest index $|G|$ and all following vertices get a smaller index. Hence, the lexicographically largest connected order-k subgraph consists of the k vertices with the highest indices. The depth-first ordering of the vertices implies a lexicographical ordering of the solutions. The initial connected subgraph of order k for the enumeration is the lexicographically largest subgraph. Next, we define the predecessor function f: The lexicographically largest subgraph is its own predecessor. For each other node A, f is defined as $f(A) \coloneqq A \setminus \{u\} \cup \{v\}$ where

Algorithm 3.7: The `RwP` algorithm.

```
1  Algorithm RwP(G, k)
2      Queue Q := ∅
3      foreach connected component C in G do
4          S := lexicographically largest solution in C
5          Q.append(S)
6          while Q ≠ ∅ do
7              S := Q.get()
8              output S
9              for vertex v ∈ S do
10                 S' := S \ {v}
11                 N := common neighborhood of connected components of S'
12                 for vertex w ∈ N do
13                     S'' := S' ∪ {w}        ▷ Now determine predecessor of S'' for
                           vertex u ∈ S'' according to ascending index sorting do
14                         S* := S'' \ {u}
15                         M := common neighborhood of connected components
                             of S*
16                         x := vertex with highest index in M
17                         if u = w and x = v then
18                             Q.append(S* ∪ {x})
```

1. $u \in A$ is the vertex with smallest index and

2. $v \notin A$ is the vertex with highest

such that $f(A)$ is connected and lexicographically smaller than A.

It was mentioned in [67] that it is necessary to consider an ordering based on DFS instead of an arbitrary ordering. The pseudocode of this algorithm can be found in Algorithm 3.7. Note that in the pseudocode analogously to `RwD` a queue Q is used maintain the nodes which still have to be processed. We added Q to the pseudocode since this matches our implementation of `RwP` (see Section 3.8.2). The difference between our implementation and the description here is that in our implementation we use BFS instead of DFS. The use of BFS is the reason of the usage of Q. This does *not* change the overall running time or the delay but requires more space. We used BFS since it can be implemented more easily.

A Slightly Improved Delay for RwP for small k. For `RwP`, we obtain an improved delay in the case $k \ll n$. To this end, we decrease the time which is needed to determine the predecessor of a solution. In the algorithm described in [67] this step needs $\mathcal{O}(k(\Delta+\log k)\min(n-k, k\Delta))$ time. We show that this step can be done in $\mathcal{O}(k^2\Delta)$ time using the new approach for computing the common neighborhood of some connected components that we proposed for `RwD`.

Proposition 3.10. *The predecessor of a connected order-k subgraph can be determined in $\mathcal{O}(k^2\Delta)$ time.*

Proof. Let S be a connected order-k subgraph. To determine the predecessor of S, we find the vertex $u \in S$ with lowest index and the vertex $v \notin S$ with highest index such that $S \setminus \{u\} \cup \{v\}$ is connected as follows: We test for each vertex w of S in increasing index order whether removing w from S gives the predecessor. Each time, we do the following: In $\mathcal{O}(k \min(k, \Delta))$ time we determine the connected components of $G[S \setminus \{w\}]$. Afterwards, we determine the common neighborhood N of the connected components in $G[S \setminus \{w\}]$ in $\mathcal{O}(k\Delta)$ time as follows: Compute for each connected C component of $G[S \setminus \{w\}]$ the set of vertices N_C that are from $V \setminus S$ and have at least one neighbor in C. Then, intersect all of these sets in $O(k\Delta)$ time; the resulting set is N. Finally, pick the vertex u with highest index in N. The set $(S \cup \{u\}) \setminus \{w\}$ is the predecessor of S. The overall running time is $\mathcal{O}(k^2\Delta)$ since we may have to consider up to k possibilities for w. □

The bottleneck in `RwP` is the predecessor check in Lines 13–18 since computing the common neighbors N of each connected component of S' (Line 11) can be done in $\mathcal{O}(\min(n-k, k\Delta))$ time and there are k possibilities for S'. With the new method for computing the predecessor we get the following delay, since the predecessor function is called (as discussed above) at most $k \cdot \min(n-k, k\Delta)$ times: we consider each vertex in S as a candidate for removal from S and check the common neighborhood.

Corollary 3.11. *The modified `RwP` algorithm solves* E-CISE *for any graph G with maximum degree Δ and integer k with polynomial delay $\mathcal{O}(k^3\Delta \min(n-k, k\Delta))$.*

If $k \ll n$, then $\min(n-k, k\Delta) \geq k$ and hence, our modified algorithm has a better delay than the algorithm described in [67]. Hence, combining both approaches and choosing one of the both algorithms depending on the value of parameter k leads to the following delay. Here, the space usage will be linear if DFS instead of BFS is used.

Corollary 3.12. *The modified `RwP` algorithm solves* E-CISE *for any graph G with maximum degree Δ and integer k with polynomial delay $\mathcal{O}(k^2 \min(n-k, k\Delta) \cdot \min(k\Delta, (n-k)(\Delta+\log k)))$ and space $\mathcal{O}(n+m)$.*

3.5 Polynomial Delay for Enumerating Connected Induced Subgraphs of Size at most k

We now consider B-CISE, the problem of enumerating all connected subgraphs of order *at most* k. Recall that Avis and Fukuda described the first algorithm with delay $\mathcal{O}(nm)$ for B-CISE based on the reverse search framework [11]. Recently, the `RSSP` algorithm (for an description we refer to Section A.4 in the Appendix) achieved a delay of $\mathcal{O}(n_c)$ [4], where n_c is the order of the largest connected component of G, for the special case when $k \geq n_c$, that is, when there is no size restriction. It is possible to adapt this algorithm to the case of arbitrary k which gives a delay bound of $\mathcal{O}(n_c + k) = \mathcal{O}(n_c)$ for B-CISE. We omit the details of this adaption and instead proceed to show that by adapting `Simple` or `Simple-Forward`, we obtain algorithms with delay $\mathcal{O}(k + \Delta)$ that need $\mathcal{O}(n + m)$ space.

Furthermore, Haraguchi and Nagamochi [108] presented an algorithm with delay $\mathcal{O}(\min(\ell + 1, n)n^5m)$ and space $\mathcal{O}(n^3)$ to enumerate all vertex sets that induce an ℓ-edge-connected subgraph and an algorithm with delay $\mathcal{O}(\min(\ell+1, n^{1/2})n^{\ell+4}m)$ and space $\mathcal{O}(n^{\ell+2})$ to enumerate all vertex sets that induce an ℓ-vertex-connected subgraph. Note that the special case $\ell = 1$ for both problems corresponds to B-CISE. Observe that the bounds on the delay and space of these algorithms is much worse than our bounds.

Now, we adapt `Simple` and `Simple-Forward` to solve the B-CISE problem. We have chosen these two algorithms for this task since the changes to the variant of enumerating all connected induced subgraphs of size *exactly* k is very small and the delay is optimal if $\Delta \ll k$. By abusing notation, we refer to `Simple` and `Simple-Forward` as the adapted variants for B-CISE.

Theorem 3.13. *`Enumerate` with `Simple` or `Simple-Forward` solves* B-CISE *for any graph G with maximum degree Δ with delay $\mathcal{O}(k + \Delta)$ and space $\mathcal{O}(n + m)$.*

Proof. As shown in Lemmas 3.4 and 3.7, `Simple` and `Simple-Forward` both spend $\mathcal{O}(\Delta)$ time at each search tree node before either creating the next child T_{i+1} or returning to the parent T_{i-1} of the current enumeration tree node T_i.

To achieve the claimed delay bound, we adapt each enumeration algorithm as follows: `Enumerate` chooses an arbitrary start vertex v. After the enumeration of all connected induced subgraphs of order at most k containing vertex v, vertex v and all incident edges are deleted in $\mathcal{O}(\Delta)$ time. Furthermore, we do not use the introduced pruning rules.

We output solutions for B-CISE according to the alternative output rule [218]: Consider a node T_i in the enumeration tree with subgraph set P_i. If i is odd, then

output P_i when node T_i is created. Otherwise, if i is even, then output P_i when the algorithm returns to the parent T_{i-1} of node T_i. In the following, a node T_i with i odd is called an *odd* node, and a node T_i with i even is called *even*.

To prove that this adaption leads to a delay of $\mathcal{O}(k + \Delta)$ for B-CISE, we bound the time between the output of two consecutive solutions for B-CISE. Clearly, the first solution of B-CISE is output after $\mathcal{O}(1)$ time, since the vertex v chosen by `Enumerate` is a solution and 1 is odd. Now, assume we just output a solution for B-CISE in some node T_i. We show that the next solution is output or the algorithm correctly terminates after a constant number of moves in the enumeration.

Case 1: Node T_i is odd. Then, node T_i was created directly before P_i was output.

Case 1.1: Node T_i has a further child T_{i+1}. The algorithm needs $\mathcal{O}(\Delta)$ time to construct this node. If node T_{i+1} has a further child T_{i+2}, the algorithm constructs this child in $\mathcal{O}(\Delta)$ time. Since T_{i+2} is odd, the algorithm immediately outputs the corresponding subgraph set P_{i+2} in $\mathcal{O}(k)$ time. Otherwise, node T_{i+1} has no child and we return in $\mathcal{O}(\Delta)$ time to node T_i. Since T_{i+1} is even the algorithm then outputs the subgraph set P_{i+1} in $\mathcal{O}(k)$ time.

Case 1.2: Node T_i has no further child. Hence, the algorithm returns in $\mathcal{O}(\Delta)$ time to the even node T_{i-1}. If node T_{i-1} has a further child, the algorithm constructs in $\mathcal{O}(\Delta)$ time the next child T_i' which is odd. Hence, subgraph set P_i' is output in $\mathcal{O}(k)$ time directly after the construction of T_i'. Otherwise, if node T_{i-1} has no further child, the algorithm return to its parent T_{i-2} and since node T_{i-1} is even, the subgraph set P_{i-1} is then output in $\mathcal{O}(k)$ time.

Case 2: Node T_i is even. Then, P_i is output directly before the algorithm returns to the parent T_{i-1} of T_i in $\mathcal{O}(\Delta)$ time.

Case 2.1: Node T_{i-1} has a further child T_i'. The algorithm constructs this node in $\mathcal{O}(\Delta)$ time. If T_i' has a child, the algorithm computes the first child T_{i+1}' of node T_i' in $\mathcal{O}(\Delta)$ time. Since node T_{i+1}' is odd, the algorithm immediately outputs the subgraph set P_{i+1}' in $\mathcal{O}(k)$ time. Otherwise, the even node T_i' has no children, and the algorithm outputs the subgraph set P_i' in $\mathcal{O}(k)$ time directly before returning to T_{i-1} .

Case 2.2: Node T_{i-1} has no further child. If $T_{i-1} = T_i$ then this iteration of `Enumerate` is finished and either the algorithm starts a new iteration of `Enumerate` and output the induced subgraph corresponding to the newly added vertex or ends the enumeration. Otherwise, if $i - 1 \neq 1$, the algorithm constructs in $\mathcal{O}(\Delta)$ time its even parent T_{i-2}. If T_{i-2} has a further child, the algorithm computes in $\mathcal{O}(\Delta)$ time its next child T_{i-1}'. Since T_{i-1}' is odd, the algorithm outputs P_{i-1}' in $\mathcal{O}(k)$ time. Otherwise, node T_{i-2} has no further child. Since T_{i-2} is even, the algorithm outputs the subgraph set P_{i-2} in $\mathcal{O}(k)$ time directly before returning to T_{i-3}.

In all cases, the delay between two consecutive outputs of a solution for B-CISE is $\mathcal{O}(k + \Delta)$. □

Since $\Delta < n_c$, `Simple` and `Simple-Forward` give improved delay bounds for small k and Δ while achieving the same delay bound as `RSSP` in the previously considered case $k = n_c$.

3.6 Polynomial Delay for Enumeration of Connected Edge-Induced Subgraphs

Next, we show that `Simple` and `Simple-Forward` can also be used to enumerate all connected edge-induced subgraphs of size at most or exactly k with polynomial delay. Recall that for an edge set $F \subseteq E$ the edge-induced subgraph of F in $G = (V, E)$ is defined as the subgraph of G with edge set F and all vertices having at least one endpoint in F. Also, recall that the size of the edge-induced subgraph of F is $|F|$, the number of edges. Formally, we study the following two problems.

Exact Connected Edge-Induced Subgraph Enumeration (E-CEISE)

Input: An undirected graph $G = (V, E)$ and an integer k.
Task: Enumerate all connected edge-induced subgraphs of size exactly k of G.

Bounded Connected Edge-Induced Subgraph Enumeration (B-CEISE)

Input: An undirected graph $G = (V, E)$ and an integer k.
Task: Enumerate all connected edge-induced subgraphs of size at most k of G.

Slightly different problems were studied by Haraguchi and Nagamochi [108]. They presented an algorithm with delay $\mathcal{O}(\min(\ell + 1, n)m^6)$ and space $\mathcal{O}(mn^2)$ to enumerate all edge sets that induce an *ℓ-edge-connected subgraph* and an algorithm with delay $\mathcal{O}(\min(\ell + 1, n^{1/2})m^{\ell+4})$ and space $\mathcal{O}(m^{\ell+1}n)$ to enumerate all edge sets that induce an *ℓ-vertex-connected subgraph*. Note that the special case $\ell = 1$ for both problems corresponds to B-CEISE. Observe that the guarantees on the delay and space of these algorithms is much worse than our guarantees. Also note that here they ask for an edge set and in the work mentioned in Section 3.5 they ask for a vertex set.

Recently, the RASMA algorithm was introduced to solve B-CEISE for the special case when $k \geq m_c$, that is, when there is no size restriction [202]. Here, m_c is the maximal number of edges of any connected component of G. Furthermore, Salem et al. [202] showed that RASMA achieves a delay of $\mathcal{O}(m_c)$ for solving B-CEISE. It is possible to adapt this algorithm to the case of arbitrary k which gives a delay bound of $\mathcal{O}(m_c + k) = \mathcal{O}(m_c)$ for B-CEISE if $k \geq m_c$. We omit the details of this adaption and instead proceed to show that by adapting Simple or Simple-Forward, we obtain algorithms with delay $\mathcal{O}(k + \Delta)$ that need $\mathcal{O}(n + m)$ space for B-CEISE. Furthermore, we show that both algorithms also solve E-CEISE with delay $\mathcal{O}(k^2\Delta)$.

Now, we adapt Simple and Simple-Forward for E-CEISE and B-CEISE. The simple idea is to adapt both algorithms by using edges instead of vertices. More precisely, the subgraph set and the extension set consist of edges instead of vertices. Furthermore, for an edge uv we denote its *neighborhood*, the set of all edges with exactly one endpoint in $\{u, v\}$ by $N(uv)$. Now, we use the algorithms Simple and Simple-Forward as described for vertex sets but now with edge sets. In other words, we essentially solve E-CISE and B-CISE on the line graph of G. The line graph consists of a vertex for each edge and two vertices in the line graph are adjacent if the two corresponding edges share a common endpoint in G.

Note that Lemma 3.4 and Lemma 3.7 still give a time bound of $\mathcal{O}(\Delta)$ per node in the enumeration tree since each edge has at most 2Δ neighbors. Also Lemma 3.2 which removes connected components of size less than k needs to be adapted. For this, note that the removal of any edge increases the number of connected components by at most 1. Hence, the proof of Lemma 3.2 can be adapted in such a way that all connected components with less than k edges are removed and that the overall running time is $\mathcal{O}(k^2)$. Thus, from Theorem 3.5, Corollary 3.8, and Theorem 3.13 we obtain the following results.

Corollary 3.14. *Enumerate with Simple or Simple-Forward solves* E-CEISE *for any graph G with maximum degree Δ with delay $\mathcal{O}(k^2\Delta)$ and space $\mathcal{O}(n + m)$.*

Corollary 3.15. *Enumerate with Simple or Simple-Forward solves* B-CEISE *for any graph G with maximum degree Δ with delay $\mathcal{O}(k + \Delta)$ and space $\mathcal{O}(n + m)$.*

Since $\Delta \leq m_c$, Simple and Simple-Forward achieve the same delay bound as RASMA [202] in the previously considered case $k = m_c$.

3.7 Polynomial Delay for Enumeration of Connected Subgraphs

Next, we show that `Simple` and `Simple-Forward` can also be used to enumerate *all* connected , not only induced, subgraphs of size at most or exactly k with polynomial delay. Recall that a graph $G' \coloneqq (V', E')$ is a subgraph of G if $V' \subseteq V$, $E' \subseteq E$. We study the following two problems.

Exact Connected Subgraph Enumeration (E-CSE)
Input: An undirected graph $G = (V, E)$ and an integer k.
Task: Enumerate all connected subgraphs of size exactly k of G.

Bounded Connected Subgraph Enumeration (B-CSE)
Input: An undirected graph $G = (V, E)$ and an integer k.
Task: Enumerate all connected subgraphs of size at most k of G.

These problems are motivated by several applications. One example is the search in semantic web data by using only keywords instead of structured queries [68]. In this application, one is interested in all connected subgraphs not only induced ones.

We are not aware of any algorithms studying these problems with respect to its delay or to output sensitivity. Here, we show how `Simple` and `Simple-Forward` can be adapted to achieve polynomial delay for both problems.

The general idea is as follows: We use `Simple` and `Simple-Forward` to enumerate all connected induced subgraphs of size exactly (at most) k. Whenever we enumerate an induced subgraph $G[S]$ with `Simple` or `Simple-Forward`, we next enumerate all spanning subgraphs of $G[S]$. Since each enumerated subgraph is spanning, each subgraph is in particular connected. We will ensure that this second step does not increase the delay too much by relying on the alternative output rule [218].

To do this second enumeration step, we show that given a connected graph, we can enumerate all spanning subgraphs with polynomial delay.

Lemma 3.16. *Let G be a connected graph. Then, all spanning subgraphs of G can be enumerate with delay $\mathcal{O}(m)$ and space $\mathcal{O}(m)$.*

Proof. In the following, we call a subgraph of G containing a spanning subgraph of G a *solution*. We use a binary enumeration tree in which the root of the enumeration tree and each left child T_{left} of any node T correspond to a solution. In this enumeration tree we do not only remove edges from G to discover all solutions, we also mark some edges which cannot be removed anymore. For an enumeration tree node T, we

denote by G_T the corresponding subgraph. The basic idea for an enumeration tree node T is to choose an unmarked edge e of G_T which is not a bridge in G_T. In the left child T_{left} of T we remove e from G_T and in the right child T_{right} we mark e. The right child is necessary to save the information that edge e was already deleted once to not enumerate some solutions multiple times. This process to create children also explains why not each node in the enumeration tree corresponds to a solution: The edge set G_T of node T is identical to the edge set $G_{T_{\text{right}}}$ of the right child T_{right} of T. If we would enumerate the subgraphs corresponding to each node we would thus enumerate some subgraphs multiple times.

Next, we explain how we create the children T_{left} and T_{right} of a node T. First, we compute with Tarjan's algorithm [214] all bridges of G_T in $\mathcal{O}(n+m) = \mathcal{O}(m)$ time. Next, we mark all not already marked bridges in $\mathcal{O}(m)$ time. If, each edge is marked, no further branching is necessary. Otherwise, we check whether there is an unmarked edge e in $\mathcal{O}(n+m)$ time. If yes, we know that e is in at least one cycle. Then, in the left child of T we remove e from G_T and in the right child of T we mark e. If T is a left child, we can output the corresponding subgraph in $\mathcal{O}(n+m) = \mathcal{O}(m)$ time. Thus, in $\mathcal{O}(m)$ time we can create the next child of T.

To restore the parent, we need to reverse these operations. This is done using the following data structures: We use one stack in which we store which edge was removed or marked, respectively, by the branching procedure. Furthermore, we use another stack to store all bridges which got marked by Tarjan's algorithm. Clearly, since each edge can be removed or marked at most once and in each enumeration tree node we remove or mark at least one edge, for both stacks we need $\mathcal{O}(m)$ space. Now, we can use these stacks to restore the parent T' of a node T as follows: First, we unmark all edges which got marked with Tarjan's algorithm at T with the second stack. Second, we insert or unmark the edge which was removed or marked in T' to create T with the first stack. Clearly, this can be done in $\mathcal{O}(m)$ time. Thus, these two stacks allow us to restore the parent node of any node in the numeration tree in $\mathcal{O}(m)$ time.

Next, we argue that our algorithm outputs each solution.

Claim 1. Each solution is output exactly once.

Proof of Claim. For any node T in the enumeration tree let $R_T \coloneqq E(G) \setminus E(G_T)$ be the set of edges removed from G in node T and let $M_T \subseteq E(G_T)$ be the set of marked edges in G_T.

We show inductively that our algorithm outputs each solution H. Let $F_H \coloneqq E(G) \setminus E(H)$. In our induction we show that there exists a sequence $T_1, T_2, \ldots, T_\ell$ of enumeration tree nodes with G_{T_1} corresponding to G, T_{j+1} being a child of T_j for

each $j \in [\ell - 1]$, and G_{T_ℓ} corresponding to H such that $R_{T_j} \subseteq F_H$ and $M_{T_j} \subseteq E(H)$ for each $j \in [\ell]$. Showing this statement in sufficient to verify that each solution is output exactly once since each solution H is uniquely defined by $E(H)$ and F_H.

It remains to verify the existence of this sequence of enumeration tree nodes. Clearly, in the root the two invariants are fulfilled since no edge is marked or removed. Next, assume that $R_{T_i} \subseteq F_H$ and $M_{T_i} \subseteq E(H)$ for each $i \in [j]$. We now verify that there exists a child T_{j+1} of T_j fulfilling $R_{T_{j+1}} \subseteq F_H$ and $M_{T_{j+1}} \subseteq E(H)$. If $G_{T_{j+1}}$ corresponds to H we have verified that H is output. In the following, we assume that $G_{T_{j+1}}$ does not correspond to H. Let B_{j+1} denote the set of bridges detected by Tarjan's algorithm in node T_{j+1}. Observe that $B_{j+1} \subseteq E(H)$ since otherwise H would not be connected, a contradiction. Since each bridge in B_{j+1} is then marked, the invariants are fulfilled. Next, an unmarked edge e is chosen. If $e \in F_H$, we consider the left child of T_{j+1} and observe that the invariants are fulfilled since $e \in R_{T_{j+1}}$. Otherwise, if $e \notin F_H$, we consider the right child of T_{j+1} and observe that the invariants are fulfilled since $e \in M_{T_{j+1}}$. Hence, we have verified that H gets enumerated.

We conclude that each solution is enumerated exactly once. ■

It remains to bound the delay of our algorithm. For this, we rely on the alternative output rule [218]. We cannot apply the alternative output rule directly since in this technique it is assumed that each node in the enumeration tree corresponds to a solution. In our setting, only the root and the left children correspond to solutions. Recall that the right children are necessary to keep track of the removed edge in the sibling to circumvent the multiple enumeration of solutions. We adapt the alternative output rule as follows: Output the subgraph corresponding to the root immediately after creating the root. Consider a node T_{left} in the enumeration tree with the corresponding subgraph $G_{T_{\text{left}}}$. If the depth of T_{left} is odd, then output $G_{T_{\text{left}}}$ when T_{left} is created. Otherwise, if the depth of T_{left} is even, then output $G_{T_{\text{left}}}$ when the algorithm returns to the parent T of node T_{left}. Now, the proof that the delay is $\mathcal{O}(m)$, that is, after a constant number of moves in the enumeration tree the next solution is output or the algorithm terminates, is similar to the proof of Theorem 3.13 in which we used the alternative output rule to show a delay of $\mathcal{O}(k+\Delta)$ for B-CISE. For this reason we skip this part of the proof. □

Now, we can solve E-CSE with polynomial delay as follows: By Theorem 3.5 and by Corollary 3.8 `Simple` and `Simple-Forward` can be implemented in such a way that all connected induced subgraphs of size exactly k can be enumerated with delay $\mathcal{O}(k^2\Delta)$ and $\mathcal{O}(n + m)$ space. Now, whenever a connected induced subgraph $G[S]$ of size exactly k is enumerated by `Simple` or `Simple-Forward`, we use the algorithm of Lemma 3.16 to enumerate all spanning subgraphs of $G[S]$ with

delay $\mathcal{O}(\min(k \cdot \min(k, \Delta), m))$ and space $\mathcal{O}(\min(k \cdot \min(k, \Delta), m)$ since the size of $E(G[S])$ is bounded by $k \cdot \min(k, \Delta)$. Since each connected induced subgraph of size exactly k has a unique vertex set S and the algorithm of Lemma 3.16 enumerates all spanning subgraphs of $G[S]$, we conclude that no connected subgraph of size exactly k is enumerated twice. Furthermore, since each connected subgraph H of size has a unique vertex set $V(H)$, at some point `Simple` or `Simple-Forward` enumerates $V(H)$ and since the algorithm of Lemma 3.16 is correct, H is enumerated. In other words, each solution is enumerates at least once.

B-CSE can be solved similar: By Theorem 3.13 `Simple` and `Simple-Forward` can be adapted in such a way to enumerate the *vertex sets* S of all connected induced subgraphs of size at most k with delay $\mathcal{O}(k + \Delta)$ and $\mathcal{O}(n + m)$ space. Recall that for B-CISE we focused on the enumeration of vertex sets instead of the induced subgraphs since otherwise the delay is dominated by the output of the solution which is $\mathcal{O}(k \cdot \min(k, \Delta))$. Afterwards, analogously to our algorithm for E-CSE, whenever a connected induced subgraph $G[S]$ of size exactly k is enumerated by `Simple` or `Simple-Forward`, we use the algorithm of Lemma 3.16 to enumerate all spanning subgraphs of $G[S]$. Since the details are identical to E-CSE, we omit them.

Hence, we obtain the following.

Corollary 3.17. *For any graph G with maximum degree Δ,* E-CSE *can be solved with delay $\mathcal{O}(k^2\Delta)$ and space $\mathcal{O}(n + m)$.*

Corollary 3.18. *For any graph G with maximum degree Δ,* B-CSE *can be solved with delay $\mathcal{O}(\min(k \cdot \min(k, \Delta), m))$ and space $\mathcal{O}(n + m)$.*

At first sight, it is very surprising that our delay of $\mathcal{O}(\min(k \cdot \min(k, \Delta), m))$ for B-CSE is worse than our delay of $\mathcal{O}(k + \Delta)$ for B-CISE. The reason for this is that for B-CISE we focused on the enumeration of vertex sets since otherwise the delay is dominated by the output of the solution which is $\mathcal{O}(k \cdot \min(k, \Delta))$. Since for fixed $G[S]$ there might exist several spanning subgraphs, for B-CSE it is necessary to enumerate also the corresponding edge sets. Also, our delay for B-CSE cannot be improved: A graph with k vertices may have up to $k \cdot \min(k, \Delta)$ edges.

Observe that if we aim to enumerate all connected subgraphs of a given graph G we may set k to n and one solves B-CSE. The algorithm of Corollary 3.18 then gives a delay of $\mathcal{O}(n + m) = \mathcal{O}(m)$. Also note that this problem can also be solved by using the algorithm of Corollary 3.15 by setting k to m to enumerate all edge-induced subgraphs of size at most m which gives the same delay of $\mathcal{O}(m)$. In other words, these two different algorithms can be used to solve the same task.

3.8 An Experimental Comparison

We now present an experimental comparison of `Simple`, `Pivot`, `Simple-Forward`, `Exgen`, `Kavosh`, `RwD`, `RwP`, and `BDDE` for E-CISE and B-CISE. For a detailed description of `Exgen`, `Kavosh`, and `BDDE`, refer to the appendix (Chapter A).

3.8.1 Experimental Setup

We implemented all algorithms[1] in Python 3.6.8 using the graph data structure *igraph*; the core modules of igraph are written in C.[2] Each experiment was performed on a single thread of an Intel(R) Xeon(R) Silver 4116 CPU with 2.1 GHz, 24 CPUs and 128 GB RAM. For the `BDDE` algorithm, we used NetworkX (`https://networkx.github.io/`) as graph library for building the enumeration tree. This choice is due to the fact that the graph modification operations of igraph are inefficient. The reported running times include the time needed to write the output to the hard drive.

As benchmark data set we used 30 sparse social, biological, and technical networks obtained from the Network Repository [200], KONECT [153], and the 10th DIMACS challenge [12]. We group the real-world instances into three subsets of size 10: small networks with $n < 500$, medium-size networks with $500 \leq n < 5000$ and large networks with $n \geq 5000$. An overview of the instance properties and names is given in Table 3.1.

In addition, we performed experiments on random instances generated in the $G(n, p)$ model where n is the number of vertices and each edge is present with probability p. We generated one instance for each $n \in \{100, 200, \ldots, 1000\}$ and $p \in \{0.1, 0.2\}$.

For each real-world network and each random instance, we considered each $k \in \{3, 4, \ldots, 10\}$ and $k \in \{n_c - 1, n_c - 2, n_c - 3\}$ where n_c is the order of the largest connected component in the graph. For each instance, we set a running time threshold of 10 minutes.

3.8.2 Implementation Details

Recall that the algorithms `Exgen`, `Kavosh`, `Simple` and `Simple-Forward` make use of the sets P_i, S_i, and F_i at each node T_i. To speed up the enumeration for large k

[1] The source code of our implementation is available at `https://www.uni-marburg.de/en/fb12/research-groups/algorith/enucon.zip`

[2] `http://igraph.org/python/`

Table 3.1: Networks used for our experiments.

Size	Name	n	m
Small	moreno-zebra	27	111
	ucidata-zachary	34	78
	contiguous-usa	49	107
	dolphins	62	159
	ca-sandi-auths	86	124
	adjnoun_adjacency	112	425
	arenas-jazz	198	2 742
	inf-USAir97	332	2 126
	ca-netscience	379	914
	bio-celegans	453	2 025
Medium	bio-diseasome	516	1 188
	soc-wiki-Vote	889	2 914
	arenas-email	1 133	5 451
	inf-euroroad	1 174	1 417
	bio-yeast	1 458	1 948
	ca-CSphd	1 882	1 740
	soc-hamsterster	2 426	16 630
	inf-openflights	2 939	15 677
	ca-GrQc	4 158	13 422
	inf-power	4 941	6 594
Large	soc-advogato	6 541	51 127
	bio-dmela	7 393	25 569
	ca-HepPh	11 204	117 619
	ca-AstroPh	17 903	196 972
	soc-brightkite	56 739	212 945
	coAuthorsCiteseer	227 320	814 134
	coAuthorsDBLP	299 067	977 676
	coPapersCiteseer	434 102	16 036 720
	soc-twitter-follows	404 719	713 319
	coPapersDBLP	540 486	15 245 729

for E-CISE, we implemented the following pruning rule for these four algorithms making use of the sets P_i and S_i: We save the order of each connected component of G. Let T_i be a node in an enumeration tree of one of these four algorithms. Let C denote the connected component of G containing all vertices in P_i. To avoid some unnecessary recursions, we check if $|C| - |F_i| < k$. If yes, we return in T_i to its

parent T_{i-1}. The correctness of this pruning rule follows by the fact that for each subsequent child T_j we have $F_i \subseteq F_j$ and adding all remaining possible vertices of C to P_i is not sufficient to obtain a solution containing k vertices.

Lemma 3.19. *Let T_i be a node in an enumeration tree of* ***Exgen****,* ***Kavosh****,* ***Simple*** *and* ***Simple-Forward****. Let P_i be the corresponding subgraph set in a connected component C of G and let F_i be the corresponding set of forbidden vertices. If $|C|-|F_i| < k$, then no subsequent recursive call in node T_i leads to an interesting leaf.*

In the following we refer to this rule as the *k-component* rule. While this rule gives a speed-up in practice it does *not* improve the delay of these algorithms. We now give some further details on how we implemented the algorithms in Python.

We implemented all three variants of `Pivot` (`Pivot`, the version of `Pivot` without active vertices, and `Simple-Forward`). Preliminary experiments revealed that `Simple-Forward` is the fastest of these three variants. Hence, we only consider `Simple-Forward` in the following. Furthermore, we implemented `Exgen`, `Kavosh`, `Simple` and `Simple-Forward` recursively and iteratively. Preliminary experiments showed that for each k the iterative variants of the algorithms are at least a factor of two faster than the recursive variants. This factor increases for large k. Hence, we only compare the iterative variants. Furthermore, preliminary experiments showed that `Kavosh` outperforms `Exgen` on almost every instance. Hence, we excluded EXGEN from our plots. Each of `Kavosh`, `Simple` and `Simple-Forward` is implemented with and without the k-component rule, and the corresponding pruning rules. In other words, each of these four algorithms has four versions:

1. The *Plain* version (no pruning rule and no k-component rule)
2. The *Pruning* version (with pruning rule and no k-component rule)
3. The *Component* version (no pruning rule but with k-component rule)
4. The *Full* version (with pruning rule and with k-component rule).

Preliminary experiments also showed that our new implementations of `RwD` and `RwP` outperform the existing ones by roughly 20%. Thus, we only consider the new versions in our plots. Next, we specify the implementation details of the specific algorithms.

Enumerate. We did not implement the removal of the vertex v after the call of `Enum-Algo` directly because *igraph* was relatively inefficient with respect to graph

modifications. Because of the same argument, we also did not implement the algorithm of Lemma 3.2. Instead we assign an index to every vertex and process the vertices in descending index order. Then, in the call of `Enum-Algo` where we enumerate all solutions containing v, we remove all vertices with higher index than v when constructing the set of neighbors of any vertex.

Simple. The set P is implemented as a list. When creating a new node T of the enumeration tree in `Simple`, we add the new vertex u to this list; when returning from T to its parent, we remove u from the list. The extension set is implemented as described in Lemma 3.4. We use two additional arrays. The first array B is used for applying the pruning rule described in Lemma 3.3, that is, it is used for implementing the variable hasIntLeaf. When we create a new child T_{i+1} of a node T_i the values $B[i]$ and $B[i+1]$ are set to 0. If an interesting leaf is detected, each entry of B is set to 1. If a boring leaf is detected, the last vertex from the subgraph set is removed, until a node T_j with $B[j] = 1$ is reached. In the second array we save the orders of the connected components of the graph G to test the k-component rule.

Simple-Forward. We implemented the old variant of `Pivot` described in Algorithm 3.3, the adapted variant of `Pivot` described in Algorithm 3.4 and the new variant which we refer to as `Simple-Forward`. For `Simple-Forward`, we implemented the data structures described in Lemma 3.7. As for `Simple`, we have an additional array B to save the result of testing the pruning rule and an additional array to save the orders of the connected components of the graph G to test the k-component rule.

Kavosh. We implemented `Kavosh` as described in Algorithm A.2. Note that X is the set of exclusive neighbors of P. To compute each $M \subseteq X$ for a fixed set X we used *itertools*. The original implementation used the *revolving door ordering* [152, 127]. Similar to `Simple` and `Simple-Forward` we have an additional array B to save the result of testing the pruning rule described in Proposition A.3 (the idea of this pruning rule is very similar to the idea for the pruning rules of `Simple` and `Simple-Forward`) and and an additional array to save the orders of the connected components of the graph G to test the k-component rule.

RwD. We implemented another method to determine the common neighborhood N of the connected components of S'. Instead of checking for each vertex of $V \setminus S$ whether it has at least one neighbor in each connected component of $G[S']$, we compute for each connected C component of $G[S']$ the set of vertices N_C that are from $V \setminus S$ and have at least one neighbor in C. Then, we intersect all of these

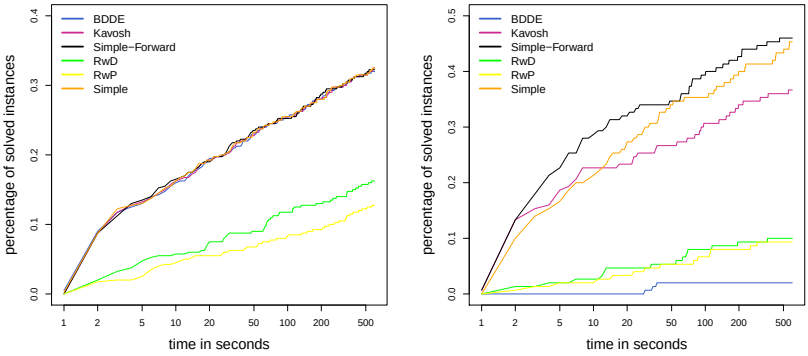

Figure 3.2: Comparison for E-CISE. Left: Comparison for $k \in \{3, \ldots, 10\}$. Right: Comparison for $k \in \{n_c - 1, n_c - 2, n_c - 3\}$ where n_c is the order of the largest connected component in the graph.

sets, the resulting set is N. We computed the connected components of $G[S']$ with a union-find structure with path compression and union-by rank. This does not change the worst-case delay of `RwD`.

RwP. The original algorithm used DFS to find all solutions for E-CISE. To achieve the claimed delay it was necessary to distinguish between nodes of odd and even depth. We implemented this algorithm with BFS. In other words, we use a queue to save subgraphs that are discovered but not processed. Hence, our implemented algorithm has a space bound of $\mathcal{O}(m + n + k|\mathcal{G}|)$, instead of $\mathcal{O}(n + m)$. Similarly to the `RwD` algorithm, we determine the connected components by a union-find structure with path compression and union-by rank and the common neighbors of these components are computed similarly.

3.8.3 Results for E-CISE

First, we consider our experimental results for $k \in \{3, \ldots, 10\}$. The Plain version of `Simple` was slightly faster than the other three versions of `Simple` which had almost the same running times. For `Simple-Forward` and `Kavosh` we obtained similar

Table 3.2: Average running times for E-CISE on instances that are solved by all four algorithms `BDDE`, `Kavosh`, `Simple`, and `Simple-Forward` for small k, that is, $k \in \{3, \ldots, 10\}$.

Category	`BDDE`	`Kavosh`	`Simple`	`Simple-Forward`
Small	40.9	39.3	37.5	34.2
Medium	91.4	87.7	86.2	78.5
Large	151.3	151.9	153.9	149.4

results: The Plain version is slightly faster than the other three versions. Overall we observe that for small k the Plain versions of the algorithms are the fastest.

In each plot only the fastest of the four versions (Plain, Pruning, Component, or Full) of `Simple`, `Simple-Forward` and `Kavosh` is plotted. The left part of Figure 3.2 shows the result for $k \in \{3, \ldots, 10\}$. `RwD` is five times faster than `RwP`. Furthermore, all instances solved by `RwD` within the time limit of 600 seconds, were solved by the other four algorithms within 15 seconds. In other words, `RwD` is 40 times slower than the other four algorithms. The differences of the running times for `BDDE`, `Kavosh`, `Simple` and `Simple-Forward` were small: `Simple-Forward` is slightly faster than the other three algorithms. `Simple` solved 130 out of 400 instances, one more than `Simple-Forward`.

The average running time of a solved instance for each algorithm (`BDDE`, `Kavosh`, `Simple`, and `Simple-Forward`) can be found in Table 3.2. `RwD` and `RwP` are not included in Table 3.2 since they are significantly slower than the other four algorithms. On average, `Simple-Forward` is the fastest algorithm in all three categories. Hence, for small k, one should use the plain version of `Simple-Forward`.

Second, we consider our experimental results for $k \in \{n_c - 1, n_c - 2, n_c - 3\}$ where n_c is the order of the largest connected component in the graph. The Plain versions of `Kavosh`, `Simple`, and `Simple-Forward` and `BDDE` solve only three out of 150 instances. Furthermore, many memory errors occurred in `BDDE`. This is not surprising since `BDDE` stores huge parts of the enumeration tree to copy branches which are used again. `RwD` has no memory errors since the cutoff time of 600 seconds was so small that `RwD` only outputted a few solutions.

For `Kavosh` we observed the following: The Pruning version is much faster than the Plain version (no concrete speed-up factor is possible since the Plain version only solved three instances). The Component version is roughly 20 times faster than the Pruning version. Finally, the Full version is roughly 30% faster than then Component version.

The right part of Figure 3.2 shows the result for $k \in \{n_c - 1, n_c - 2, n_c - 3\}$

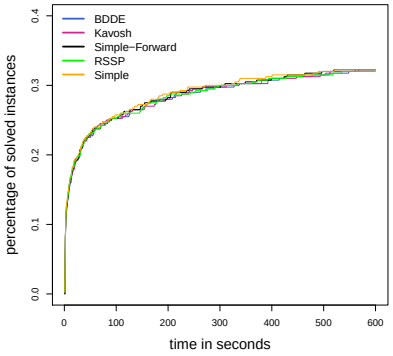

Figure 3.3: Comparison for B-CISE for $k \in \{3, \ldots, 10\}$.

where n_c is the order of the largest connected component in the graph.

`Simple` with the pruning rule and without the k-component rule is much faster than the plain version. Similar to `Kavosh` we cannot estimate a speed-up factor since the plain version solved only very few instances. `Simple` with the k-component rule and with the pruning rule is slightly faster than `Simple` with the k-component rule and without the pruning rule. Both variants are roughly 30 times faster than `Simple` only with the pruning rule. We obtained similar results for `Simple-Forward` as for `Simple`. In our experiments the k-component rule gives a much higher speedup than the corresponding pruning rules since k was at least $n_c - 3$. To conclude: For large k the versions of the algorithms with the pruning rule and with the k-component rule are the fastest.

`RwD` is roughly two times faster than `RwP`. All instances solved by `RwD` are solved by `Kavosh` in less than two seconds. `Simple` is roughly three times faster than `Kavosh` and `Simple-Forward` is roughly two times faster than `Simple`. `Simple-Forward` solved 69 out of 150 instances. Hence, for large k, one should use `Simple-Forward` with the pruning rule and the k-component rule.

Table 3.3: Average running times for B-CISE on instances that are solved by all algorithms for small k.

Category	BDDE	Kavosh	RSSP	Simple	Simple-Forward
Small	45.9	45.4	44.6	39.7	44.9
Medium	76.5	75.3	62.4	68.6	70.7
Large	141.6	145.4	159.0	141.7	140.6

3.8.4 Results for B-CISE

We tested the plain versions of `Kavosh`, `Simple` and `Simple-Forward` for B-CISE. We compare these three algorithms with `BDDE` [167] and `RSSP` [4]. Here, we do not compare with `RwD` or `RwP` since these algorithms are designed specifically for E-CISE. Figure 3.3 shows our results for $k \in \{3, \ldots, 10\}$. For B-CISE we did no experiments for $k \in \{n_c-1, n_c-2, n_c-3\}$ where n_c is the order of the largest connected component in the graph since the number of solutions is monotonically increasing for each k and gets too huge soon.

`BDDE` is slightly slower than `Kavosh`, which is slightly slower than `RSSP`. Next, `Simple-Forward` is slightly faster than `RSSP` and `Simple` is slightly faster than `Simple-Forward`. Overall, `Simple` is roughly 20% faster than `BDDE`. `Simple` solved 129 out of 400 instances. The average running time of a solved instance for each algorithm in our comparison can be found in Table 3.3. `Simple` is the fastest algorithm for small instances, `RSSP` is the fastest algorithm for medium instances and `Simple-Forward` is the fastest algorithm for large instances.

Our experiments showed that the running times between all these algorithms for B-CISE do not differ too much. But since in our experiments `Simple` was the fastest, one should use this algorithm.

3.9 Conclusion

We studied several variants of subgraph enumeration problems. On the theoretical side, we improved the current best delay delay for E-CISE presented by Elbassioni [67] to $\mathcal{O}(k^2\Delta)$, using only linear space (Theorem 3.5 and Corollary 3.8). Furthermore, we also provided polynomial-delay algorithms for B-CISE (Theorem 3.13). For the special case $k = n_c$ our delay matches the previous best delay proven by Alokshiya et al. [4]. Also, we showed that our algorithms can be used to enumerate all connected edge-induced subgraphs with exactly (at most) k edges (Corollaries 3.14

and 3.15). For the special case $k = m_c$ our delay matches the previous best delay proved by Salem et al. [202]. Finally, we showed that also the enumeration of all subgraphs with exactly (at most) k vertices can be done with polynomial delay (Corollaries 3.17 and 3.18). On the practical side, we provided a comprehensive study of Python implementations for E-CISE and B-CISE. For B-CISE and also for E-CISE and small k we showed that the running-time of all search-tree based algorithms is comparable. This is mainly because there are many solutions and the time to output them dominates the running time. In contrast, for E-CISE and large k the running time of `Simple-Forward` is significantly smaller than of previous algorithms (`RwD` and `RwP`).

In Chapter 4 we use the implementations of `Simple`, `Simple-Forward`, and `Kavosh` as the basic enumeration methods in an efficient generic algorithm for fixed-cardinality optimization problems in graphs. For this application, `Simple-Forward` turned out to be the most efficient implementation.

Concerning future work, a major open task is to improve the delay bounds for the problems studied in this chapter. For this, E-CSE, however, could be a good starting problem. We provided a delay of $\mathcal{O}(k^2\Delta)$ for E-CSE (Corollary 3.17) which is the same delay we provided for E-CISE. This is because in our current algorithm for E-CSE uses our algorithm for E-CISE as a subroutine: In our current approach we enumerate all induced connected subgraphs of size k and each time the vertex set S of an induced connected subgraph is enumerated, we then enumerate all subgraphs of $G[S]$ which contain a spanning subgraph of $G[S]$. Since the number of solution for E-CSE is larger than the number of solutions for E-CISE one would expect a smaller delay for E-CSE than for E-CISE. To obtain a smaller delay a more direct approach could be helpful, that is, an approach that does not use the enumeration of all induced connected subgraphs as a subroutine.

Smaller delays than $\mathcal{O}(k^2\Delta)$ are also desirable for E-CISE and E-CEISE. For both problems, in our opinion, the main bottleneck is the number of enumeration tree nodes which need to be visited until we are guaranteed to find the next solution. Currently, we bounded this number by k^2. We think that a bound of k for this number is within reach: In our arguments we currently do *not* exploit the fact that if the extension set is big, that is, if it has size $k + \ell$, then node T_{k-1} has at least ℓ children which correspond to solutions.

Also, it is interesting to study the `Kavosh` algorithm [127] we used in our experiments in terms of bounded delay. We already introduced a pruning rule for `Kavosh` (see Proposition A.3). The idea of `Kavosh` is to try all possible extensions decreasingly in their size. Our pruning rule end this process if no extension for some size leads to a new solution. However, this pruning rule is *not* sufficient to obtain

polynomial delay: Consider the star graph with one vertex v of degree $n-1$ and assume v is added in the root of the enumeration tree. After trying all subsets of size $k-1$ of $N(v)$, the algorithm tries to add each subset of size $k-2$, none of which gives a solution. The number of these subsets is $\binom{n-1}{k-2}$ which is not polynomial if k is not a constant. Hence, further pruning rules are necessary to obtain polynomial delay. Another algorithm which we used in our experiments is `BDDE` [167]. For `BDDE` polynomial delay seems impossible without massively changing the algorithm: The basic idea of `BDDE` is to enumerate all connected vertex sets S' with $S' \subset S$ before the connected vertex set S is enumerated. Clearly, the number of connected subsets of S is exponential in $|S|$.

Furthermore, it would be interesting to study the algorithm of Ferreira [79] from the viewpoint of bounded delay. Recall, that in this algorithm a binary search tree whose nodes represent connected sets S of G is used together with a certificate to guarantee that each solution is enumerated exactly once. Since the depth of the enumeration tree in this algorithm is not bounded by k, one would need a different analysis to prove delay bounds for this algorithm. Furthermore, it is interesting implement algorithm of Ferreira [79] and to compare its running time with the running times of the algorithms studied in the chapter.

A further interesting route to obtain better enumeration algorithms could be to replace the maximum degree Δ by provably smaller parameters such as the h-index or the degeneracy d of the graph. Such algorithms need more advanced data structures since already one scan of the neighborhood of a vertex may take $\Omega(\Delta)$ time. Since h and d are smaller parameters than Δ not to many expensive neighborhood scans are possible anymore. One idea is to not scan the whole neighborhood of a vertex v contained in a current vertex subset S at once, instead one could check only one neighbor w of v, extend S by w and only after all solutions containing w are enumerated the next neighbor of v is checked. One issue one has to overcome in this idea is that up to Δ many neighbors of v may have already been visited by other vertices in S. In other words, without some additional arguments also this idea leads to an algorithm in which $\mathcal{O}(\Delta)$ time is spend in each enumeration tree node.

Another interesting route for further research is to study the problem of counting the number of connected (induced) subgraphs of size exactly k. It was shown that counting the number of connected induced subgraphs of size exactly k is #W[1]-hard with respect to k [121]. Furthermore, it was shown that the number of all connected subgraphs of size k for $k \leq 5$ can be counted in linear time in graphs with constant degeneracy [19, 184, 196]. Recall that all connected induced subgraphs of size k can be enumerated in $\mathcal{O}((e(\Delta-1))^{(k-1)} \cdot (\Delta+k) \cdot (n/k))$ time [150]. Can this running time be significantly reduced for the counting problem, for example by replacing Δ

by the h-index?

Finally, it would be interesting to determine systematically for which restrictions on the connected induced subgraphs that we aim to enumerate one may achieve better delay bounds. For example, in `FixCon` (see Chapter 4) we sometimes do not want to enumerate all connected induced subgraphs of size k. Consider the Densest k-Subgraph problem. There, one searches for a vertex set S of size exactly k such that $G[S]$ has the maximum number of edges of all these vertex sets. For this problem, we do *not* have to consider vertices u in the enumeration for which another vertex v exists such that $N(u) \subseteq N[v]$. Is it also possible to enumerate all connected vertex sets of size k which fulfill a property like this with polynomial delay?

Chapter 4

FixCon: A Generic Solver for Fixed-Cardinality Subgraph Problems

In many classic graph problems, one is interested in finding a set S of exactly k vertices maximizing (or minimizing) a specific objective function $\mathrm{val}_G(S)$ in a given graph G [35, 38, 39, 150]. For example, in Densest k-Subgraph we aim to maximize the number of edges with both endpoint in S [74, 77, 150] (also see Chapter 8). In contrast, in Sparest k-Subgraph we aim to minimize the number of edges with both endpoint in S [106, 107, 224] (also see Chapter 8). In Max $(k, n-k)$-Cut one aims to find a set S of size k maximizing the number of edges having exactly one endpoint in S[38, 60, 87] (also see Chapter 8). Another example is Maximal Triangles in which we aim to maximize the number of triangles in $G[S]$, the subgraph induced by S [217]. Also, recognition problems fit into this framework. For a property Π we set $\mathrm{val}_G(S) = 1$ if $S \in \Pi$ and $\mathrm{val}_G(S) = 0$ if $S \notin \Pi$. One example is Acyclic Subgraph; in this problem one aims to find a set S which induces an acyclic subgraph (a forest) [94]. All these problems are so-called fixed cardinality optimization problems and can be defined generally as follows.

Fixed-Cardinality Optimization (FCO)

Input: An undirected simple graph $G = (V, E)$, an objective function $\mathrm{val}_G \colon \mathcal{G} \to \mathbb{R}$, and an integer k.

Task: Find a set $S \subseteq V$ of size k such that S maximizes $\mathrm{val}_G(G[S])$ under this condition.

Herein, $\mathcal{G}$ is the set of all undirected simple graphs. Bruglieri et al. [35] give

a systematic survey about special cases of FCO where they present results on the computational complexity of some special cases and discuss some methods to solve them. Furthermore, the survey of Ehrgott and Hamacher [66] discusses some results on the computational complexity of some fixed-cardinality optimization problems, for example NP-hardness of k-Cardinality Tree. Also, they discuss many results on approximation algorithms and integer linear programs (ILPs). Cai [38] studied the parameterized complexity of many FCO problems. For example, Cai [38] showed that Max $(k, n-k)$-Cut is W[1]-hard with respect to k; in Chapter 8 we study the parameterized complexity of Max $(k, n-k)$-Cut in further detail. Since FCO contains Max $(k, n-k)$-Cut as a special case, FCO is thus NP-hard and also W[1]-hard with respect to k. Despite its general nature, there exist algorithmic results for FCO: Cai et al. [39] presented a framework, called the random separation method, to obtain FPT-algorithms for the parameter $k + \Delta$ for a wide range of special cases. The random separation method is a variation of the color-coding technique [5] with the aim of coloring the solution in one color and all neighbors of the solution in another color.

Often, maximizing or minimizing the objective function val_G is not sufficient. Instead, one has the additional constraint that the sought vertex set S is connected. Such a connectivity constraint is particularly well-motivated in network design. For example, in the design of wildlife corridors, a necessary requirement is to allow animals to reach each part of the network [48]. Similarly, connectivity is also desirable in the construction of wildlife reserves to prevent fragmentation [32]. Also, in oil field leasing, it is important to detect valuable connected parts of a geographic area [105]. More generally, in facility layout problems it is important to create connected facilities [80]. Furthermore, in the analysis of protein networks it is important to identify connected sets of vertices [6]. Finding connected vertex sets is also an important task in the design of efficient algorithms: For example, in local-search based algorithms for Vertex Cover, one searches for a connected vertex set of fixed size to improve the solution [128]. Adding the connectivity constraint to FCO leads to the following problem.

Connected Fixed-Cardinality Optimization (CFCO)

Input: An undirected simple graph $G = (V, E)$, an objective function $\text{val}_G \colon \mathcal{G} \to \mathbb{R}$, and an integer k.

Task: Find a set $S \subseteq V$ of size k such that $G[S]$ is connected and S maximizes $\text{val}_G(G[S])$ under these conditions.

Similar to FCO, CFCO contains Clique is NP-hard and W[1]-hard with respect to k. On the positive side, the random separation method gives a randomized

algorithm for CFCO with running time $\mathcal{O}(2^{(\Delta+1)k} \cdot |V|^{\mathcal{O}(1)} \cdot T_{\text{val}_G}(k))$ where $T_{\text{val}_G}(k)$ is the time needed for evaluating val_G on graphs of order k and Δ is the maximum degree of G [39]. Later, this result was improved: Komusiewicz and Sorge [150] presented an algorithm that solves CFCO in $\mathcal{O}((e(\Delta-1))^k \cdot |V|^{\mathcal{O}(1)} \cdot T_{\text{val}_G}(k))$ time. Furthermore, Maxwell et al. [167] proposed the algorithm `BDDE` to enumerate *all* connected induced subgraphs H that for a given *hereditary* objective function val_G and threshold t, fulfill $\text{val}_G(H) > t$. Here, a function val_G is denoted as *hereditary* if $\text{val}_G(S) \geq t$ then also $\text{val}_G(S') \geq t$ for each $S' \subseteq S$. According to Lemma A.5 the running time of `BDDE` is bounded by $\mathcal{O}((e(\Delta-1))^k \cdot k \cdot \Delta \cdot n)$.

In this chapter, we engineer an algorithm for CFCO. Before we do this, let us first inspect the algorithm of Komusiewicz and Sorge with running time $\mathcal{O}((e(\Delta-1))^k \cdot |V|^{\mathcal{O}(1)} \cdot T_{\text{val}_G}(k))$ for CFCO [150] in greater detail: This algorithm consists of two steps. First, enumerate all connected induced subgraphs of order at most k, and second, evaluate val_G for the subgraphs $G[S]$ of order *exactly* k, keeping the best subgraph $G[S]$ and output S after the enumeration has finished. An implementation of this algorithm was developed for the special case when the task is to decide whether G contains an order-k connected μ-clique, where a graph with k vertices is a μ-clique if it has at least $\mu \cdot \binom{k}{2}$ edges [151]. As a proof of concept, this implementation showed that enumeration of connected subgraphs can be a useful algorithmic approach for special cases of CFCO.

The current state of such algorithms is the following: one has to provide an implementation for each problem which includes the development of new reduction and pruning rules. This makes it unlikely that these algorithms will find widespread use in real-world applications. To circumvent these issues, our aim in this chapter is to obtain rules which prevent us from enumerating all connected induced subgraphs of order at most k. For the correctness of these rules, we need the assumption that val_G depends only on the isomorphism class of $G[S]$. In other words, for any two isomorphic graphs H and H' we have $f(H) = f(H')$. Roughly speaking, the function value only depends on the structure of $G[S]$. Many problems like Clique fulfill this property, but not every CFCO problem: An example is Maximum Partial Vertex Cover. In this problem, we ask for a vertex set S of size k such that as many edges as possible have at least one endpoint in S. Note that we study the parameterized complexity of Maximum Partial Vertex Cover in Chapter 8.

In contrast to the existing enumeration-based solver for finding an order k connected μ-clique [151], integer programming and SAT solving have proved to be extremely useful tools for solving NP-hard problems. One reason for this is that SAT and ILP solvers have been engineered over decades like Gurobi or CPLEX and, as a result, can solve many real-world instances very quickly. Another reason is that,

due to the generic nature of SAT and ILP, one may formulate many combinatorial optimization problems rather easily as a SAT or ILP problem.

In this chapter, we aim to lift enumeration-based algorithms from specific to generic applications. We develop the algorithmic tool `FixCon` in which the user needs to program only the objective function val_G. The user may furthermore provide some properties of the objective function val_G that will then be exploited by generic pruning rules that restrict the search space for the enumeration algorithm. The pruning rules assume only the correctness of the provided properties and otherwise treat val_G as a black box. In addition, the user may implement problem-specific pruning rules. Since the theoretical running time guarantees for CFCO are good only in the case of small k [39, 150], we focus on engineering `FixCon` for $k \leq 20$. We did not choose the random separation method presented by Cai et al. [39] as basis for our implementation since it has worse running time guarantees and since it seems hard to exploit properties of the objective functions during the algorithm. We do not compare with `BDDE` [167] since the enumeration problem is much harder: our algorithm may stop immediately after finding one optimal solution; in the enumeration problem, one must output all solutions and thus continue the search. From another perspective, our implementation is more general than `BDDE` from Maxwell et al. [167], as val_G does not need to be hereditary in `FixCon`.

Our contribution. We implement three variants of the generic enumeration-based algorithm of Komusiewicz and Sorge [150] based on different algorithms to enumerate all connected induced subgraphs described in Chapter 3. The basic idea is that each enumeration tree node T in `FixCon` corresponds to a connected set of vertices C. In each child of T a neighbor of C is added to the connected set in `FixCon`. We then provide three generic pruning rules that help decreasing the size of the search tree used by the enumeration algorithm. To develop these rules, we note and make use of two properties of the objective function val_G: vertex-addition bounds (that limit the change of the objective value when a vertex is added to the graph) and edge-monotonicity (which means that adding an edge to a graph does not decrease its objective value). In addition, we decrease the search space size by identifying vertices with the same neighborhoods in the input graph (called *twins*). Furthermore, for edge-monotone properties we extend the notion of twins to prune the search space further. Finally, in the universal graph rule for a given vertex set C we test each possibility to extend $G[C]$ to a graph of order k. The universal graph rule and the rule based on twins are valid for any function val_G that assigns the same value to graphs from the same isomorphism class. Moreover, we provide generic heuristics that provide `FixCon` with lower bounds for the value of the optimal solution. Finally,

for some problems, we provide new problem-specific reduction rules.

In our implementation, we only focus on maximizing the objective function val_G. Clearly, one could also ask to minimize val_G. This can be easily incorporated in our framework by maximizing the function $-\mathrm{val}_G$.

We analyze the algorithms and the effect of the generic and problem-specific pruning rules for eight example problems and compare them with ILP formulations for these problems. In a nutshell, we show that the best version of our algorithm can solve the majority of the benchmark instances within 600 seconds per instance. Moreover, we outperform the ILP formulations in terms of number of solved instances.

Our source code is available at `https://www.uni-marburg.de/en/fb12/research-groups/algorith/software/fixcon`.

4.1 Example Problems

In this section, we describe eight example objective functions that we will consider as input in the CFCO instances. For each problem, we only specify the objective function val_G. All eight example objective functions fall in one of the following categories:

1. Finding dense cohesive subgraphs,
2. finding sparse subgraphs, and
3. finding subgraphs were we pose restrictions on the degree of a vertex in the solution S.

Clearly, not each CFCO problems falls into one of these three categories. For example, the problem of finding an induced subgraph $G[S]$ on k vertices which maximizes the number of even cycles does not fit into one of these categories.

Dense subgraph problems. The first two problems have applications in finding cohesive subgraphs [74, 77]. In the first problem, we aim to maximize the number of edges in the subgraph.

Densest k-Subgraph:

$$\mathrm{val}_G(H) \coloneqq |E(H)|.$$

In the second, problem we aim to find a graph in which the minimum degree is large.

Max-Min-Degree Subgraph:

$$\mathrm{val}_G(H) \coloneqq \min_{v \in V(H)} \{|N_H(v)|\}.$$

For both problems, cliques of order k give the best objective values for all graphs of order k. A further problem in this direction would be to minimize the diameter of the induced subgraphs. We did not include this problem in the comparison since for $k \leq 20$, finding solutions with diameter 2 (2-clubs) is trivial in our instances since the induced subgraph of any vertex v of degree at least $k-1$ together with exactly $k-1$ neighbors Z has diameter 2 if and only if there exists two non-adjacent vertices u and w in Z. Thus, the problem is essentially only to decide whether there is a clique on k vertices.

Sparse subgraph problems. The next four problems are concerned with finding sparse subgraphs. The first is essentially the opposite of Max-Min-Degree, that is, we aim to find a subgraph with a minimal maximum degree; we formulate it as a maximization problem.

Min-Max-Degree Subgraph:

$$\mathrm{val}_G(H) \coloneqq - \max_{v \in V(H)} \{|N_H(v)|\}.$$

The next two problems are essentially recognition problems, formulated as maximization problems. We may find induced subgraphs that are trees by solving the following problem.

Acyclic Subgraph:

$$\mathrm{val}_G(H) \coloneqq \begin{cases} 1 & H \text{ is a tree,} \\ 0 & \text{otherwise.} \end{cases}$$

In a similar fashion, we may look for connected induced subgraphs that contain no triangle.

Triangle-Free Subgraph:

$$\mathrm{val}_G(H) \coloneqq \begin{cases} 1 & H \text{ contains no triangle,} \\ 0 & \text{otherwise.} \end{cases}$$

The final problem from this group is to search for a subgraph that has a large diameter.

Maximum-Diameter Subgraph:

$$\mathrm{val}_G(H) \coloneqq \max_{u,\,v \in V(H)} \mathrm{dist}_H(u, v).$$

For all four problems, a path on k vertices gives the best objective value for all graphs of order k. Note that for Acyclic Subgraph and Triangle-Free Subgraph the set of optimal solutions is larger.

Degree-constrained subgraph problems. In the final two problems, we consider two variants of restricting vertex degrees. Again, these problems are basically recognition problems, formulated as maximization problems.

r-Regular Subgraph:

$$\mathrm{val}_G(H) \coloneqq \begin{cases} 1 & H \text{ is } r\text{-regular}, \\ 0 & \text{otherwise}. \end{cases}$$

In the experiments, we use $r = 3$ for even k and $r = 4$ for odd k, since with these values, there exist connected graphs on k vertices fulfilling the properties for all $k \in \{4, \ldots, 20\}$.

(α, β)-Degree-Constrained Subgraph:

$$\mathrm{val}_G(H) \coloneqq \begin{cases} 1 & H \text{ has minimum degree at least } \alpha \\ & \quad \text{and maximum degree at most } \beta, \\ 0 & \text{otherwise}. \end{cases}$$

We set $\alpha = 3$ and $\beta = 5$ in the experiments, since with these values, there exist connected graphs on k vertices fulfilling the properties for all $k \in \{4, \ldots, 20\}$.

The source code of our implementation of the eight above-mentioned objective functions is shown in Section B.1 in the appendix.

4.2 Experimental Setup

Each experiment was performed on a single thread of an Intel(R) Xeon(R) Silver 4116 CPU with 2.1 GHz, 24 CPUs and 128 GB RAM running Python 3.6.8. `FixCon` is implemented in Python 3.6.8. and uses igraph with the python-igraph interface (`http://igraph.org/python/`) as the graph data structure. We performed experiments on 40 real-world networks from Konect [153], the DIMACS Challenge on Graph Clustering and Partitioning [12], and the Network Repository [200]; 10 networks are sparse and small (less than 500 vertices), 10 are sparse and have medium size (500 – 5 000 vertices), 10 are sparse and large (between 5 000 and 500 000 vertices), and 10 are dense (less than 100 vertices and up to 200 000 vertices). The instance names and some of their properties are shown in Table B.1 in the appendix. To also study instances with a different structure, we built 20 random instances in the $G_{n,p}$ model with $n \in \{100, 200, \ldots, 1000\}$ and $p \in \{0.1, 0.2\}$. We set a timeout of 600 seconds per instance. The time for reading the input is not included in the running time. For the ILP, we do include the time for passing the initial set of constraints. For each graph, we built an instance for each of the eight problems and each $k \in \{4, \ldots, 20\}$. We call k *small* if $k \leq 10$ and *large* if $11 \leq k \leq 20$.

We add improvements to the algorithms and evaluate their effect one by one. In any section, we compare the version with the new improvement and all previous improvements to the variant that has all previous improvements but not the new one.

4.3 Enumeration Algorithms

We consider three different algorithms for enumerating connected induced subgraphs of order k which we investigated in Chapter 3. More precisely, we use `Simple` (Algorithm 3.2), `Simple-Forward` (Algorithm 3.5), and `Kavosh` (Algorithm A.2). To avoid confusion, we refer to `Simple-Forward` as `Pivot` in this chapter[1]. We adapted these three algorithms to CFCO as follows: instead of output the connected set C, we evaluate $\mathrm{val}_G(G[C])$ and save C if it gives the current-best objective value.

Recall that in all of the three algorithms, there is one main algorithm loop (Algorithm 3.1) which considers, in some order, each vertex $v \in V(G)$, enumerates all connected induced subgraphs of order k containing v, and then removes v from G. Also, recall that all three algorithms are search tree algorithms in which each node of

[1]The name *Pivot* is used since *Simple-Forward* is an efficient implementation of *Pivot* (also see Section 3.3)

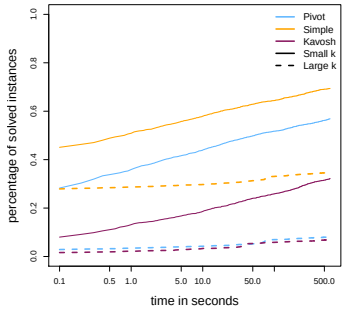

Figure 4.1: Comparison of the plain version of the three algorithms.

the search tree is associated with a set C of vertices that induces a connected graph, called a *connected set* in the following. In each search tree node with $|C| < k$, we branch into the possibilities to add vertices of $N(C)$.

In the following, we partition the connected set C into the set P which can have further neighbors and the set Q which can have no further neighbors. In other words, each vertex added to the connected set C must have at least one neighbor in P and no neighbor in Q. Recall that such a partition is explicitly given in `Pivot` and `Kavosh`. In `Simple` this is not the case, that is, each vertex in C may have further neighbors. Note that in all three algorithms, a vertex v added to the connected set C may be added to Q if $N(v)$ is already contained in $N[C]$ since no vertex w in $V(G)\setminus(C\cup\{v\})$ is a neighbor of v.

In the plain version of our implementation, we use the following easy way to prune the search space: we stop the search once a solution that is obviously optimal has been found. To this end, users may provide `FixCon` with a *global* upper bound for the value of val_G on graphs of order k. For many problems such a bound is easy to determine. For Densest k-Subgraph the global upper bound is $\binom{k}{2}$, for Max-Min-Degree Subgraph it is $k-1$. For Min-Max-Degree Subgraph, the global upper bound is -2, as a connected graph on at least four vertices has at least one vertex that has two or more neighbors. For Maximum-Diameter Subgraph, the global upper bound is $k-1$, this bound is met only by the path on k vertices. Finally, for the remaining problems which all describe graph properties, the best objective value is 1 which can be seen directly from the definition.

The the running times of the plain version of the three enumeration algorithms is shown in Figure 4.1. `Kavosh` is substantially slower than `Pivot` which is again

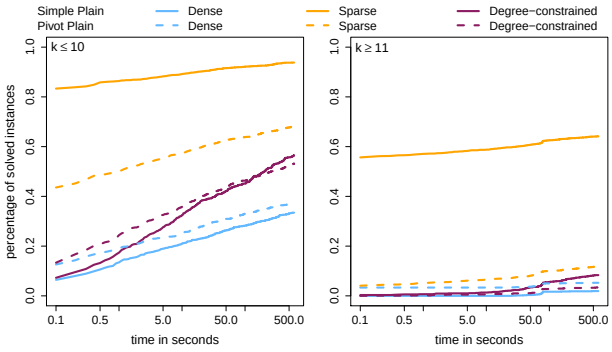

Figure 4.2: Comparison of the running times of the plain version of `Simple` and `Pivot` for the three problem categories.

substantially slower than `Simple`. In all further versions of the algorithms that we tested, `Kavosh` was slower than the other two algorithms. This is due to the structure of the enumeration tree in `Kavosh` which has, on average, smaller *depth* and larger *breadth* than the search trees of `Simple` and `Pivot`. In other words `Kavosh` has many leaves (corresponding to subgraphs of order k) and few inner nodes (corresponding to subgraphs of order less than k). Since the further improvements are rooted in the idea to prune the search tree at some inner node, `Kavosh` benefits less from these improvements. Hence, to improve the presentation, `Kavosh` is excluded from further experiments.

Figure 4.2 shows the performance of the plain versions of `Simple` and `Pivot` separated by categories. `Pivot` is only faster in the dense category. The main observations are that the sparse problems are the easiest for the plain algorithm version, followed by the degree-constrained problems, and the dense problems which are the hardest. `Simple` solved 69 % of the instances with small k and 34% of the instances with large k. Thus, as expected, the instances with large k are much harder than those with small k.

4.4 Pruning Rules

The plain algorithm only prunes the search tree if the graph contains a solution that meets the global upper bound which is often not the case. Thus, we propose

three additional pruning rules that help to further decrease the number of search tree nodes. In these pruning rules, we use the current connected set C to obtain an upper bound on the value of $\mathrm{val}_G(G[S])$ for all $S \supsetneq C$. In the formulation of the pruning rules, we denote $G[C]$ by H_C for brevity.

4.4.1 Vertex-based Upper Bounds

To allow for a computation of an upper bound, we first consider the following property of objective functions val_G.

Definition 4.1. An objective function val_G is *vertex-addition-bounded by value x*, if for every graph H and all graphs H^* that are obtained by adding some vertex to H and making this vertex adjacent to some subset of $V(H)$, we have $\mathrm{val}_G(H^*) \leq \mathrm{val}_G(H) + x$.

We may now use the following rule.

Pruning Rule 4.1 (*Vertex-Addition Rule*)**.** *Let C be the current connected set, let $|C| = k-\ell$, let val_G be vertex-addition-bounded by x, and let z denote the objective value of the current best solution. If $\mathrm{val}_G(H_C) + \ell \cdot x \leq z$, then return to the parent node in the search tree.*

We can use the *Vertex-Addition Rule (VAR)* for all eight problems. For example, the vertex-addition bound is $k-1$ for Densest k-Subgraph and 1 for Max-Min-Degree-Subgraph. Moreover, the objective function for Triangle-Free Subgraph is vertex-addition-bounded by 0, since adding a new vertex to H_C can only introduce new triangles and never destroys existing triangles. More generally, when the aim is to find an induced subgraph fulfilling some hereditary property, then we may use an objective function that is vertex-addition-bounded by 0. For example, our objective function for Acyclic Subgraph is also vertex-addition-bounded by 0.

For Min-Max-Degree Subgraph the vertex-addition bound is 0 as adding a vertex does not decrease the maximum degree. For Maximum-Diameter Subgraph the vertex-addition bound is 1, as adding a vertex may increase the diameter by at most 1.

Finally, for the two problems from the degree-constrained category, we can adapt the definition of val_G to distinguish two cases for graphs that do not fulfill the property. For example, if a subgraph H_C contains a vertex with degree at least $r+1$, then H_C is not the subgraph of an r-regular graph. Similarly, if H_C contains a vertex with degree at least $b+1$, then H_C is not the subgraph of a graph fulfilling the constraints of (a,b)-Degree-Constrained Subgraph. By setting the objective

value for such graphs to be $-\infty$ and setting the vertex-addition bound to 1, we can model this observation rather easily.

r-Regular Subgraph:

$$\mathrm{val}_G(H) \coloneqq \begin{cases} 1 & H \text{ is } r\text{-regular,} \\ -\infty & H \text{ has a vertex } v \text{ with } |N(v)| > r, \\ 0 & \text{otherwise.} \end{cases}$$

(α, β)-Degree-Constrained Subgraph:

$$\mathrm{val}_G(H) \coloneqq \begin{cases} 1 & H \text{ has minimum degree at least } \alpha \\ & \text{and maximum degree at most } \beta, \\ -\infty & H \text{ has a vertex } v \text{ with } |N(v)| > \beta, \\ 0 & \text{otherwise.} \end{cases}$$

The results if adding the *VAR* are shown in Figure 4.3. Both algorithms benefit from a substantial speed-up when using the *VAR*. Hence, *VAR* is enabled in all following variants. The effect of *VAR* for `Pivot` and `Simple` for the three categories is shown in the bottom part of Figure 4.3. `Pivot` is now faster than `Simple` in all categories for small and large k, now solving almost all instances of the sparse category. The comparison of the plain version of `Simple` and `Pivot` with the *VAR* is shown in the top part of Figure 4.3. For small k and the dense and the degree-constrained category, *VAR* gives a speedup factor of more than 10. For instances of the degree-constrained category and large k, all instances solved by the plain version of `Simple` are now solved within 0.1 seconds by `Pivot` with *VAR*. For instances of the dense category with large k, *VAR* has almost no effect as the algorithm enumerates too many dense subgraphs of order roughly $k/2$ which are not pruned by *VAR*.

Summarizing, with *VAR* enabled, `Pivot` solved 79 % of the instances for small k and 59 % of the instances for large k.

4.4.2 Edge-Monotonicity

We call an objective function val_G *edge-monotone* if adding an edge to a graph H does not decrease the objective value $\mathrm{val}_G(H)$. For Densest k-Subgraph the function val_G is edge-monotone, since adding an edge increases the objective value by 1. Similarly, val_G is edge-monotone in Max-Min-Degree Subgraph. The other six objective functions are not edge-monotone. We use edge-monotonicity as

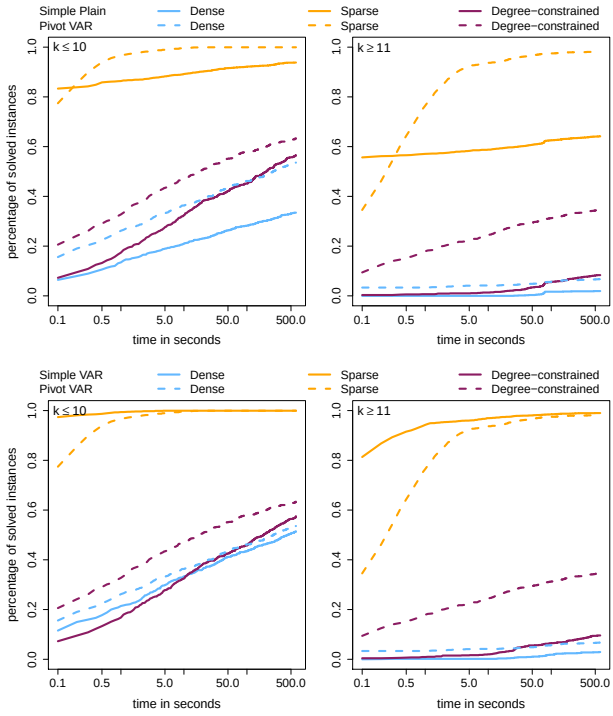

Figure 4.3: Top: Comparison of the plain version of `Simple` and `Pivot` with *VAR*. Bottom: Comparison of `Simple` and `Pivot` with *VAR*.

follows to prune the search tree. Recall that $P \subseteq C$ is the set of vertices whose neighbors can be added to extend the connected set C.

Pruning Rule 4.2 (*Clique Join Rule*)**.** *Let C be the current connected set, let $\ell = k - |C|$, and let z denote the current best objective value. Let H_{C^*} be the graph obtained from H_C by adding an ℓ-vertex clique K and making it adjacent to all vertices of $P \subseteq C$. If $\mathrm{val}_G(H_{C^*}) \leq z$, then return to the parent node in the search tree.*

The correctness of the *Clique Join Rule (CJR)* is obvious, since all connected

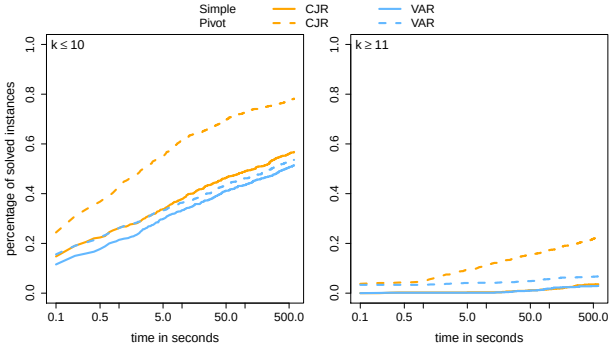

Figure 4.4: Comparison of `Pivot` and `Simple` with *VAR* and *CJR*, respectively.

induced subgraphs found in the subtree rooted at the current node are a subgraph of H_{C^*}. Consequently, they cannot achieve a better objective value than z.

The upper bound provided by the *VAR* is never better than the one provided by the *CJR*. However, the vertex-addition bound is faster to compute since we do not need to add a clique to the current subgraph H_C. Hence, we first apply the *VAR* and then the *CJR*.

Figure 4.4 shows the effect of the *CJR*. `Pivot` with the *CJR* is roughly 100 times faster than `Pivot` with only the *VAR*, the previously fastest for this category. For `Simple`, the speed-up is much smaller. We conclude that any further variants of the program should include some variant of the *CJR* when val_G is edge-monotone.

4.4.3 Universal Graphs

In contrast to the two rules above, the following pruning rule puts no restrictions on the objective function val_G. Such rules are highly desirable but it is intuitively clear that it is hard to prune the search tree when we have no knowledge about val_G.

Consider a connected set C of size $k - \ell$ at some node in the search tree and assume that z is the current best objective value. As in the other rules, if H_C can not be expanded to a connected subgraph H_{C^*} of order k such that $\mathrm{val}_G(H_{C^*}) > z$, we can discard the connected set C and hence return to the parent of the current node. To test this condition without any knowledge of val_G, we simply try each possibility to expand H_C to a connected subgraph of order k. This can be done as follows.

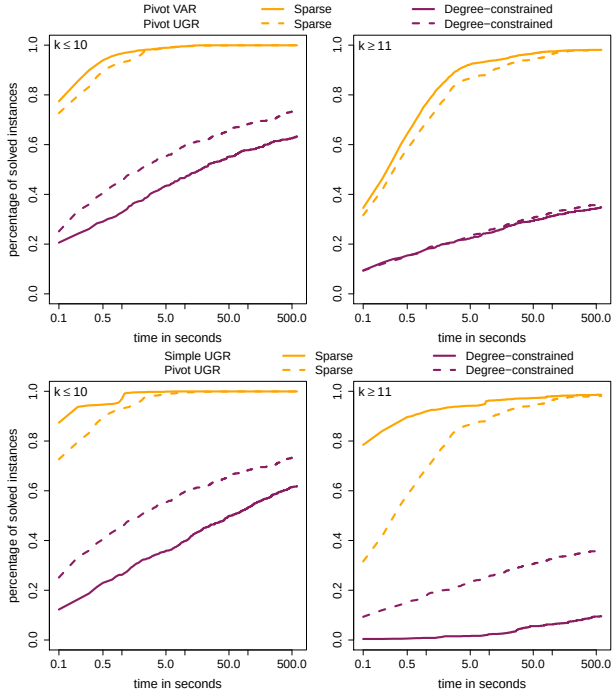

Figure 4.5: Top: Comparison of `Pivot` with the *UGR* and `Pivot` with *VAR*. Bottom: Comparison of `Simple` and `Pivot` with the *UGR*.

Pruning Rule 4.3 (*Universal Graph Rule*)**.** *For each (up to isomorphism) graph J on ℓ vertices, try each possibility of adding edges between $V(J)$ and P such that the resulting graph H_{C^*} is connected. If $\mathrm{val}_G(H_{C^*}) \leq z$ for all resulting graphs H_{C^*}, then return to the parent of the current node.*

Clearly, one may abort the pruning rule after encountering the first H_{C^*} with $\mathrm{val}_G(H_{C^*}) > z$. The *Universal Graph Rule (UGR)* is obviously correct but it is not clear when this rule will lead to an improvement in running time as the number of graphs and edge additions to consider grows exponentially in ℓ and in $|P|$. To this end, we compute the number of graphs that we generate.

Let $I(n)$ be the number of graphs (up to isomorphism) of order n. In the *UGR* we have to evaluate $y = I(\ell) \cdot 2^{\ell \cdot |P|}$ graphs. Since y can be extremely large, we apply the universal graph rule only for $\ell \leq 3$ where $I(\ell) = 2^{\ell-1}$. In addition, we compare y with an estimate of the number of search tree nodes in the subtree rooted at the current node corresponding to C.

For this estimation, we consider the size of $Y \coloneqq N(P) \setminus F$ to obtain an estimate on the number of vertices we will add to the current connected set C. More precisely, we apply the *UGR* if $2^{|P| \cdot \ell + \ell - 1} < |Y|^{\ell}$.

The effect of the *UGR* for `Pivot` is shown in the top part of Figure 4.5; the dense category is excluded from the experiments, since the *UGR* is never better than the *CJR*. For the problems of the sparse category this rule had a small negative effect. This is due the fact that `Pivot` with the *VAR* already solved almost all instances in that category. For problems of the degree-constrained category and large k, the rule had almost no effect, but for small k the rule gives a speed-up of factor 20 compared with `Pivot` with the *VAR*. Hence, we recommend to use the *UGR* for objective functions that are not edge-monotone. The running time comparison of `Simple` and `Pivot` is shown in the bottom part of Figure 4.5. `Simple` is slightly faster than `Pivot` in the sparse category but overall both variants solve the same number of instances in the sparse category. In the degree-constrained category, `Pivot` is much faster. For the future, it seems promising to fine-tune the *UGR*, for example by applying the rule if $2^{|P| \cdot \ell + \ell - 1} < c_u \cdot |Y|^{\ell}$ where c_u is a constant whose optimal value may be experimentally determined.

Summarizing, with the *CJR* for the dense category and the *UGR* for the other categories, `Pivot` solved 88 % of the instances for small k and 63 % of the instances for large k.

`Pivot` benefits more from both rules than `Simple` because the set P is smaller for `Pivot`. Recall that in `Simple` the Extension Set processed from the last to the first element and in `Pivot` the extension set is processed form the first to the last element. This distinction has the result that in some node T with the corresponding connected set C some vertices move from P to Q in `Pivot` which is not the case for `Simple`. Thus, it could be promising to consider further enumeration strategies which often lead to small P.

4.5 Neighborhood-based Reduction- and Pruning Rules

To further decrease the running times, we consider the relations between the neighborhoods of vertices in G.

4.5.1 Twin Sets

The first idea is to avoid enumerating too many graphs that are isomorphic by identifying vertices that have the same neighborhood. For this, we use the following definition. Two vertices u and v are called *true twins* if $N[u] = N[v]$ and *false twins* if $N(u) = N(v)$. If u and v are true twins or false twins, u and v are called *twins*. Two vertices u and v are true twins, then there does not exist any vertex w such that u and w are false twins, and vice versa. As a consequence, being twins is an equivalence relation. A maximal set of twins is called a *twin set*.

Before starting the enumeration algorithm, we compute all twin sets and apply the following two reduction rules.

Reduction Rule 4.1. *Let $\mathcal{C}$ be a set of true twins of G with $|\mathcal{C}| > k$. Remove $|\mathcal{C}| - k$ vertices of $\mathcal{C}$ from G.*

Reduction Rule 4.2. *Let $\mathcal{I}$ be a set of false twins of G with $|\mathcal{I}| \geq k$. Remove $|\mathcal{I}| - k + 1$ vertices of $\mathcal{I}$ from G.*

The correctness of the rules follows from the fact that each solution containing exactly k vertices can contain at most k vertices of a true twin set and at most $k-1$ vertices of a false twin set since at least one common neighbor of these false twins has be contained in the connected set to be connected.

During the enumeration, we use twin sets as follows.

Pruning Rule 4.4 (*Twin Rule*)**.** *Let C be the connected set at the current node, let F denote the current set of forbidden vertices and let v be a vertex such that the algorithm considered each solution extending $C \cup \{v\}$. Let $\mathcal{C}$ denote the twin set of v. Add all vertices of $\mathcal{C} \setminus C$ to F.*

The correctness can be seen as follows: Let z be the current best objective value. Assume there exists another connected set C' with $C \cup \{u\} \subseteq C'$ and $u \in \mathcal{C} \setminus C$ such that $\mathrm{val}_G(H_{C'}) > z$. Consider the set $C'' \coloneqq (C' \setminus \{u\}) \cup \{v\}$. Since u and v are twins, $H_{C'} \cong H_{C''}$. Since $v \in C''$, the algorithm already considered the connected set C''. Hence, $\mathrm{val}_G(H_{C''}) \leq z$, a contradiction to the assumption that $\mathrm{val}_G(H_{C'}) > z$.

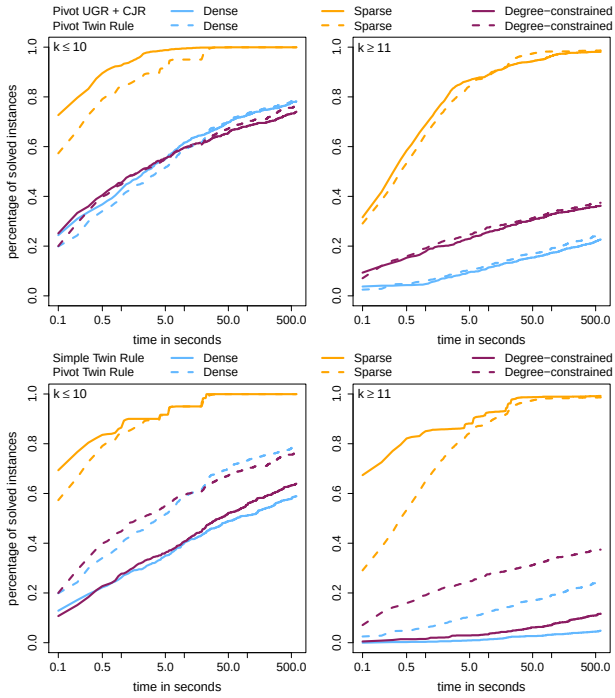

Figure 4.6: Top: Comparison of `Pivot` with the *Twin Rule* and `Pivot` with *UGR* and *CJR*. Bottom: Comparison of `Simple` and `Pivot` with the *Twin Rule*.

The effect of the *Twin Rule* for `Pivot` is shown in the top part of Figure 4.6. For running times below ten seconds, `Pivot` with *Twin Rule* is slower than `Pivot` with the *CJR* and *UGR*. This is due to the time which is needed to calculate the twin sets. In the sparse category, the *Twin Rule* gave no improvement since almost all instances can be solved within the time limit. For the dense and degree-constrained category, the *Twin Rule* gives a running time improvement of roughly factor 2 for the harder instances. Moreover, as shown in Figure 4.7, the *Twin Rule* may decrease the size of the search tree tremendously in some cases. Since the running time overhead incurred by the rule is not too high, we enable the twin rule for all three categories.

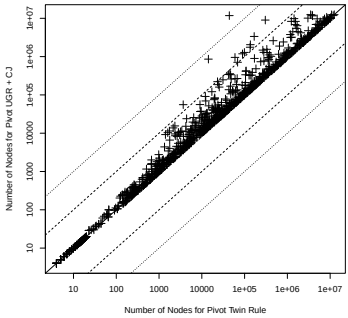

Figure 4.7: Comparison of the number of search tree nodes of `Pivot` with the *Twin Rule* and `Pivot` with *UGR* and *CJR*.

The bottom part of Figure 4.6 compares `Simple` and `Pivot` with the *Twin Rule*. In the sparse category, `Simple` is competitive. In the dense and degree-constrained category, for small k, `Pivot` is more than ten times faster than `Simple` and for large k, `Pivot` solves each instance that was solved by `Simple` within the time limit in less than one second.

4.5.2 Neighborhood Relation for Edge-Monotonicity

In the following, we provide an improvement of the Twin Rule for edge-monotone objective functions f.

Pruning Rule 4.5 (*Neighborhood Inclusion Rule*)**.** *Let C be the connected set at the current node, let F denote the current set of forbidden vertices, and let v be a vertex such that the algorithm considered each solution extending $C \cup \{v\}$. Add all vertices $u \notin C$ with $N(u) \subseteq N[v]$ to F.*

The correctness of the *Neighborhood Inclusion Rule* (*NIR*) can be seen as follows: Let z be the current best objective value. Assume there exists a connected set C' with $C \cup \{u\} \subseteq C'$, $u \notin C$ and $N(u) \subseteq N[v]$ such that $\text{val}_G(H_{C'}) > z$. Since $N(u) \subseteq N[v]$, $H_{C'}$ is a subgraph of $H_{C''}$ where $C'' \coloneqq (C' \setminus \{u\}) \cup \{v\}$. Since val_G is edge-monotone, $\text{val}_G(H_{C'}) \leq \text{val}_G(H_{C''})$. Hence, $\text{val}_G(H_{C'}) \leq z$, a contradiction to the fact that $\text{val}_G(H_{C'}) > z$.

Figure 4.8 shows the effect of the *NIR*. Similar to the *Twin Rule*, for running times below 10 seconds, `Pivot` with the *NIR* is slower than `Pivot` without it, which

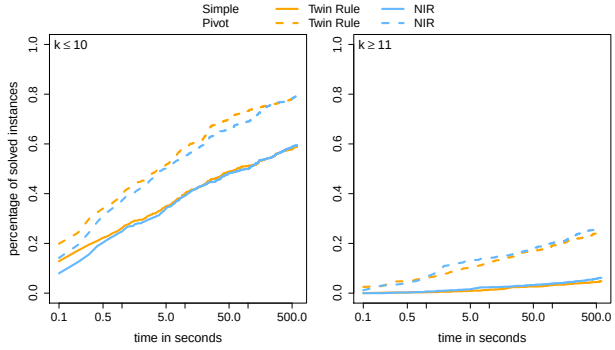

Figure 4.8: Comparison of `Simple` and `Pivot` with and without the *NIR*.

is again due to the time needed to compute the relations. For small k, the *NIR* does not improve `Pivot`. For large k, `Pivot` with *NIR* is slightly faster than `Pivot` with *Twin Rule*. Moreover, as shown in Figure 4.9, there are again some cases in which the number of search tree nodes is decreased tremendously. Hence, we enable this rule in the following. For `Simple` and small k the speedup is much better, but `Pivot` remains roughly 20 times faster than `Simple`.

Summarizing, with the *NIR* for the dense category and the *Twin Rule* for the other categories, `Pivot` solved 88 % of the instances for small k and 65 % of the instances for large k.

4.6 Heuristic Lower Bounds

We also implemented three randomized heuristics to compute good initial solutions. These provide a lower bound for the objective function val_G which will be useful for pruning the search tree. The heuristics work as follows: We have a probability measure P_0 on $V(G)$. Choose a start vertex v_1 with probability $P_0(v_1)$. For a connected set C of size $\ell < k$ we have a probability measure P_ℓ on $N(C)$, and choose a vertex v_ℓ with probability $P_\ell(v_\ell)$.

In the first heuristic, we let P_i be the uniform distribution. This heuristic is aimed to be good for objective functions where we have little knowledge.

In the second heuristic, we set $P_0(v) := |N(v)|/(2|E|)$ for each $v \in V(G)$. Furthermore, for each $1 \leq i < k$ and for each $v \in N(C)$, where C is the current connected

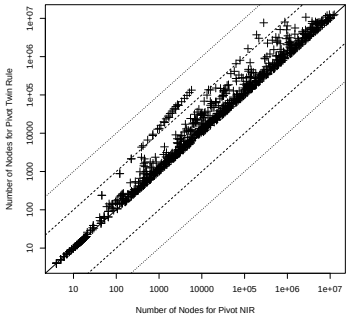

Figure 4.9: Comparison of number of search tree nodes of `Pivot` with the *Twin Rule* and `Pivot` with the *NIR*.

set, we set: $P_i(v) \coloneqq |N(v) \cap C|/R$, where $R \coloneqq \sum_{v \in N(C)} N(v) \cap C$. This heuristic is aimed to be suitable for edge-monotone objective functions.

In the third heuristic, we set $P_0(v) \coloneqq (\Delta - |N(v)|)/T$, where $T \coloneqq \sum_{v \in V(G)} (\Delta - |N(v)|)$ for each $v \in V(G)$. Furthermore, for each $1 \leq i < k$ and for each $v \in N(C)$, we set: $P_i(v) \coloneqq (\Delta - |N(v)|)/M$, where $M \coloneqq \sum_{v \in N(C)} (\Delta - |N(v)|)$. This heuristic is aimed to be suitable for problems in the sparse category.

Since instances with larger k and larger n are harder, we apply the heuristic more often, the larger n and k are. More precisely, each heuristic is applied $\log(n) \cdot k$ times. We have chosen a linear dependence on k since the subgraph size in our experiments is at most 20 and a logarithmic dependence on n since we consider networks with up to 500 000 vertices. We refer to each application as a *trial*. Let z denote the objective value of the current best solution.

While iteratively building a random subgraph in some trial, we check whether the current subgraph can still lead to a solution with objective value better than z by applying the *VAR*. If this is not the case, then we discard the current subgraph and start with the next trial.

Since the variance of the best solution found by these heuristics is huge, after each trial of each heuristic we perform local search to improve the solution quality of the found connected set C: In the local search, we consider all possibilities of obtaining a better connected set C^* by swapping a vertex $v \in C$ with a vertex $u \in V \setminus C$ as follows. For each $v \in C$, we compute the connected components $C_1, \ldots, C_\ell$ of $H_C - v$. Afterwards, we compute their common neighborhood $\mathcal{N} \coloneqq (N(C_1) \cap \ldots \cap N(C_\ell)) \setminus C$ in $G - C$. We then compute the vertex $u \in \mathcal{N}$ for which the connected set $C^* \coloneqq$

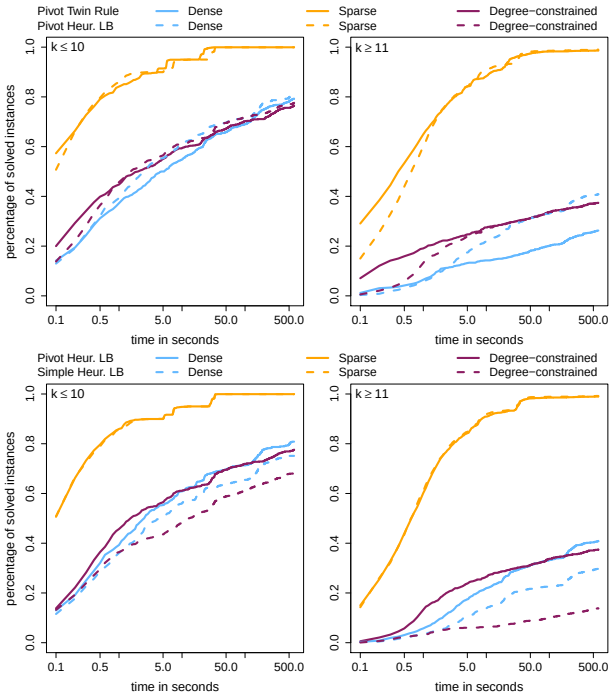

Figure 4.10: Top: Comparison of `Pivot` with and without heuristic lower bounds. Bottom: Comparison of `Simple` and `Pivot` with heuristic lower bounds.

$C \cup \{u\} \setminus \{v\}$ obtained by removing v and adding u has a maximal value $\mathrm{val}_G(H_{C^*})$. If $\mathrm{val}_G(H_{C^*}) > z$ set $C \leftarrow C^*$, $z \leftarrow \mathrm{val}_G(H_{C^*})$ and continue, otherwise we abort the trial.

The effect of the computation of the heuristic lower bounds for `Pivot` is shown in the top part of Figure 4.10. For the small instances, we can observe a small negative effect that is caused by the additional time needed to compute the lower bounds. For the large instances, we either observe no effect or a small positive effect with one exception: for the dense category and large k, the rule gives a speed-up factor of almost 100 and almost doubles the number of instances that can be solved within

the time limit. The comparison of `Pivot` and `Simple` with heuristic lower bounds is shown in the bottom of Figure 4.10. `Simple` is only competitive with `Pivot` in the sparse category.

With the greedy heuristics and all previous rules, `Pivot` solved 89 % of the instances for small k and 69 % of the instances for large k, outperforming `Simple` also in this variant.

4.7 Problem-Specific Pruning Rules

To allow for further improvement of the algorithms, we extend `FixCon` to include problem-specific pruning rules that, in addition, use not only H_C as input but also the graph G. We design such rules for the two harder categories: the dense category and the degree-constrained category.

4.7.1 Dense Category

Maximizing the Number of Edges. First, we describe the pruning rules applied for Densest k-Subgraph. The first rule is an adaptation of a known upper bound on the number of edges of any order-k graph containing the connected set C [151, 186]. We adapt this bound to our setting as follows. Let C be the current connected set, let S be any order-k solution extending C, and let $\ell \coloneqq k - |C|$ denote the number of vertices to add. We partition the edges of $G[S]$ into three subsets: the edges between vertices of C, whose number is denoted by $m(C)$ and already known, the edges between vertices of C and $S \setminus C$, whose number will be denoted by $m(C, S \setminus C)$, and the edges between vertices of $S \setminus C$, whose number will be denoted by $m(S \setminus C)$. We compute upper bounds on $m(C, S \setminus C)$ and $m(S \setminus C)$.

For bounding $m(C, S \setminus C)$, we exploit that no neighbors of vertices in $C \setminus P$ may be added. We thus first compute the set $Y \coloneqq V \setminus (C \cup N(C \setminus P) \cup F)$ of vertices that may still be added to C. For each vertex $y \in Y$, we compute $\deg_P(y) \coloneqq |N(y) \cap P|$ and $\deg_{V \setminus C}(y) \coloneqq |N(y) \cap V \setminus C|$. Now $m(C, S \setminus C) + m(S \setminus C) \leq b_0(C, P)$ where

$$b_0(C, P) \coloneqq \sum_{i=1}^{\ell} (\deg_P(y_i) + \min(\deg_{V \setminus C}(y_i), \ell - 1)/2)$$

where $\{y_1, \ldots, y_\ell\} \subseteq Y$ is the set of vertices that maximizes this sum. The second summand is divided by 2 since these edges are double counted in the sum. Moreover, if $b_0(C, P)$ is not integral, we may round down. Thus, we obtain the first improved pruning rule.

Pruning Rule 4.6. *Let C be the current connected set and let z denote the current-best objective value. If $m(C) + \lfloor b_0(C, P) \rfloor \leq z$, then return to the parent node.*

We now further refine this bound by partitioning the set of edges between vertices in $S \setminus C$ even further. To this end, let $S_1 \coloneqq (S \cap N(P)) \setminus C$ denote the vertices of $S \setminus C$ that are neighbors of P, and $S_2 \coloneqq S \setminus (C \cup S_1)$ denote the remaining vertices. We partition the edges in $S \setminus C$ into those edges inside S_1, those edges between S_1 and S_2 and those edges inside S_2. Moreover, we consider all possible sizes ℓ' of S_2; due to the connectivity constraint we have $|S_1| \geq 1$ and thus $0 \leq \ell' < \ell$. For each ℓ', we compute an upper bound of $m(S, \ell')$, defined as the maximum number of edges we may achieve by having exactly ℓ' vertices in S_2. The upper bound for $m(S)$ is then the maximum of $m(S, \ell')$ over all ℓ'. Thus, in the following, we describe how to compute bounds for $m(S, \ell')$. For fixed ℓ', we denote the numbers of the edge sets in the partition by $m(S_1, \ell')$, $m(S_1, S_2, \ell')$ and $m(S_2, \ell')$, respectively.

The number $m(S_2, \ell')$ is at most $\binom{\ell'}{2}$ since in the best case, the vertices of S_2 form a clique. To bound $m(S_1, \ell')$ we compute $\deg_{N(P)}(v) \coloneqq |N(v) \cap N(P)|$ for all vertices in $Y \cap N(P)$. To bound $m(S_1, S_2, \ell')$, we compute $\deg_{V \setminus N[C]}(v) \coloneqq |N(v) \setminus N[C]|$ for vertices in $Y \cap N(P)$. We now obtain the bound $m(C, S \setminus C, \ell') + m(S_1, \ell') + m(S_1, S_2, \ell') + m(S_2, \ell') \leq b_1(C, P, \ell')$ where

$$\begin{aligned} b_1(C, P, \ell') \coloneqq \sum_{i=1}^{\ell - \ell'} \Big(& \deg_P(v_i) \\ & + \min(\deg_{N(P)}(v_i), \ell - \ell' - 1)/2 \\ & + \min(\deg_{V \setminus N[C]}(v_i), \ell') \Big) \end{aligned}$$

where $v_1, \ldots, v_{\ell - \ell'}$ are the vertices of $Y \cap N(P)$ maximizing the sum.

Pruning Rule 4.7. *Let C be the current connected set and let z denote the current-best objective value. If $m(C) + \max_{0 \leq \ell' < \ell}(\lfloor b_1(C, P, \ell') \rfloor + \binom{\ell'}{2}) \leq z$, then return to the parent node.*

Evaluation. The top part of Figure 4.11 shows the effect of both pruning rules. The variant where Pruning Rule 4.6 is added to Heur. LB is referred to as `Old UB` and the variant where Pruning Rule 4.7 is added is referred to as `Specific UB`. `Old UB` gives a considerable improvement for small k and large k. Using in addition the new upper bound (`Specific-UB`) gives a rather small improvement for large k and has a negligible effect for small k.

Maximizing the Minimum Degree. Second, we describe the pruning rules for Max-Min-Degree-Subgraph which are simpler than the ones for Densest k-Subgraph.

Pruning Rule 4.8. *Let C be the current connected set and let z denote the current-best objective value. If*

1. *C contains a vertex with degree at most z in G, or*
2. *C contains a vertex v with at most $z - \deg_{H_C}(v)$ neighbors in Y, or*
3. *$z > \ell - 1$ and $N(P) \cap Y$ contains less than ℓ vertices v with $|N(v) \cap P| + \min(\ell - 1, |N(v) \setminus C|) > z$, or*
4. *$N(P) \cap Y$ contains no vertex with $|N(v) \cap P| + \min(\ell - 1, |N(v) \setminus C|) > z$*

then return to the parent node.

The first two cases of the pruning rule are obviously correct. For the third case, observe that if $z > \ell - 1$, only vertices with at least one neighbor in P may be added. Moreover, a vertex may only be added if his number of neighbors in P plus the number of neighbors in the remaining part of S exceeds z. If there are less than ℓ candidates to add, then there is no solution extending C. For the last case, observe that we need to add at least one vertex of $N(P) \cap Y$. If there is no suitable candidate, then we can discard the current connected set.

4.7.2 Degree-Constraint Category.

We will only describe the rules for the more general (α, β)-Degree-Constrained Subgraph since we use the same rules for r-Regular Subgraph by setting $\alpha = \beta = r$.

In the following, for each vertex of P, we let $\mathrm{dem}(v) \coloneqq \alpha - \deg_C(v)$ denote the *demand* of v, that is, the number of edges with one endpoint being v that we need to add in order to fulfill the degree constraint of v. Here, $\deg_C(v)$ is the degree of the vertex in H_C. Recall that $Q = C \setminus P$ denotes the set of vertices for which we may not add further neighbors.

Pruning Rule 4.9. *Let C be the current connected set. Return to the parent node if*

1. *Q contains a vertex of degree less than α, or*

2. C contains a vertex v with $\mathrm{dem}(v) > k - |C|$*, or*

3. C contains a vertex v that has degree less than α in G.

This rule is correct since in each case the vertex v cannot have degree at least α.

The following rule is correct, since in the cases described by the rule, every addition of a vertex to C creates at least one vertex of degree at least $\beta + 1$.

Pruning Rule 4.10. *Let C be the current connected set. If $|C| < k$ and if all vertices of P have degree β in H_C, then return to the parent node.*

In the next case, the idea is to count how many edges need to be added in order to increase the degree of every vertex of H_C above the threshold α. This number is compared with an upper bound on the number of edges that we may obtain by adding $k - |C|$ new vertices whose degree may not exceed β.

Pruning Rule 4.11. *Let C be the current connected set. Let $V_{<\alpha}$ denote the set of vertices in H_C that have degree less than α. If $\sum_{v \in V_{<\alpha}} \mathrm{dem}(v) > (k - |C|) \cdot \beta$, then return to the parent node.*

We now take a closer look at the vertices of $V(G) \setminus C$ that we may add to C. More precisely, we exploit the following observation: we may not add any vertex u that has degree less than α in G or any vertex u that has a neighbor w in C which has degree β in H_C. Thus, we may define for each vertex $v \in C$, a set of *possible neighbors*

$$N^*_G(v, C) \coloneqq \{u \in V(G) \setminus C \mid \ \deg_G(u) \geq \alpha \text{ and } \forall\, w \in C \cap N(u) \colon \deg_C(w) < \beta\}.$$

Pruning Rule 4.12. *Let C be the current connected set. If C contains a vertex v such that* $\mathrm{dem}(v) > |N^*_G(v, C)|$*, then return to the parent node.*

We now look at pairs of vertices u and v in C for which we still need to add neighbors. We exploit the following observation: the fewer the number of common neighbors of u and v in $V(G) \setminus C$, the larger is the number of vertices that we must add. If this number is too large, then C cannot be extended to a solution.

Pruning Rule 4.13. *Let C be the current connected set.*

1) If C contains two vertices u and v such that $\mathrm{dem}(v) + \mathrm{dem}(u) - \min(\beta - \deg_C(v), \beta - \deg_C(u), |N^*_G(v, C) \cap N^*_G(u, C)|) > k - |C|$*, or*

2) if C contains three vertices u, v, and w such that $N^*_G(w, C)$ is disjoint from $N^*_G(u, C)$ and from $N^*_G(v, C)$ and $\mathrm{dem}(v) + \mathrm{dem}(u) + \mathrm{dem}(w) - \min\{\beta - \deg_C(v), \beta - \deg_C(u), |N^*_G(v, C) \cap N^*_G(u, C)|\} > k - |C|$*, then return to the parent node.*

The correctness of the first part of the rule can be seen as follows: In order to fulfill the degree constraints for v and u, we need to add $\text{dem}(v) + \text{dem}(u)$ edges with an endpoint in u or v. Vertices that are common neighbors of u and v add two such edges, all other vertices add only one such edge. Hence, we may write the number of vertices to add as the number of neighbors of u plus the number of neighbors of v minus the number of common neighbors of u and v. Since the number of common neighbors of u and v that we may add is at most $\min\{\beta - \deg_C(v), \beta - \deg_C(u), |N^*_G(v, C) \cap N^*_G(u, C)|\}$, the left hand side thus gives a lower bound on the number of vertices that we need to add to increase the degree of u and v sufficiently. If this lower bound exceeds $k - |C|$, the number of vertices that we may still add, then there is no solution C' that extends C.

For the second part, we simply consider a third vertex w if the set of possible neighbors of w is disjoint from those of v and u. The arguments for the correctness are completely analogous. Since the second part of the rule is costly in terms of running time (we have to consider all triples of vertices in C), we apply this part only if $\text{dem}(v) + \text{dem}(w) - \min\{\beta - \deg_C(v), \beta - \deg_C(u), |N^*_G(v, C) \cap N^*_G(u, C)|\} > (k - |C|)/2$. The rationale behind this condition is that it is unlikely that the second part of the rule will be successful if the condition is not met.

The final rule extends Rule 4.13 to larger subsets of vertices in C. Instead of trying all possibilities of such sets, we greedily compute a set of vertices in C for which we still need to add neighbors and which have disjoint possible neighborhoods.

Pruning Rule 4.14. *Let C be the current connected set. If C has a subset A of vertices such that for all vertices $u, v \in A$ we have $N^*_G(v, C) \cap N^*_G(u, C) = \emptyset$ and $\sum_{v \in A} \text{dem}(v) \geq k - |C|$, then return to the parent node.*

As stated above, the set A is computed greedily. More precisely, we consider the vertices of C in some order and add the first vertex v in $C \setminus A$ with $\text{dem}(v) > 0$ whose possible neighborhood is disjoint from all possible neighborhoods of A.

Evaluation. The overall effect of these pruning rules is shown in the bottom of Figure 4.11. There is a substantial speed-up for the dense category, particularly for large k, and a tremendous speed-up for the degree-constrained category for small and large k.

4.8 A Comparison with ILP formulations

To compare `FixCon` with some competitor, we developed ILP formulations for all eight problems. As ILP solver, we used Gurobi version 8.01 with the Python interface.

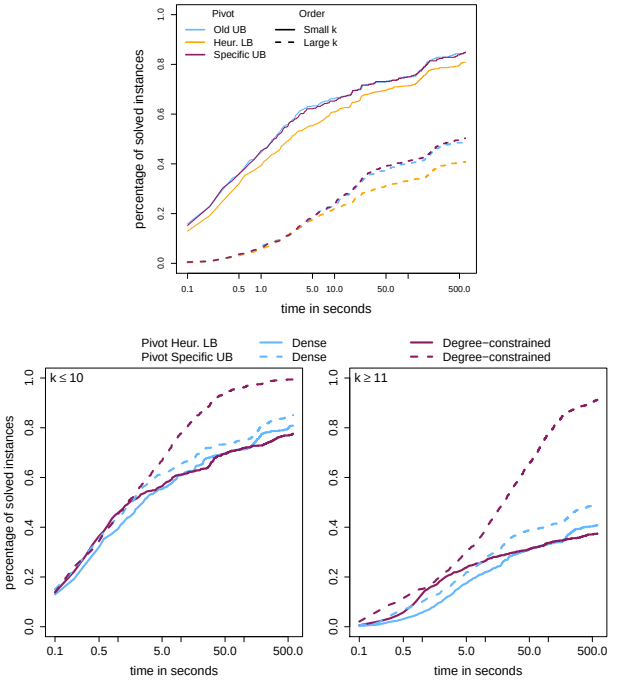

Figure 4.11: Comparison of `Pivot` with and without problem-specific pruning rules. Top: The two rules for Densest k-Subgraph. Bottom: Comparison for the dense and degree-constrained category.

In all eight formulations we have binary variables x_v for each vertex $v \in V$ and the constraint $\sum_{v \in V} x_v = k$ which ensures that k vertices are selected. As proposed by Althaus et al. [6] we use lazy constraints to ensure the connectivity of the solution: If at some node of the search tree, we have a disconnected solution S, then we add

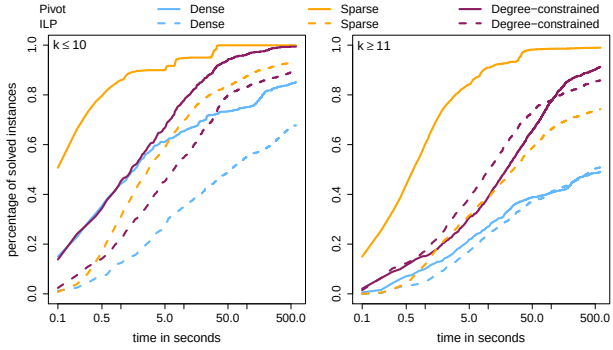

Figure 4.12: Comparison of the ILP with the version of `Pivot` containing all improvements.

the following constraint for each connected component C of $G[S]$ in a callback [2]

$$\sum_{v \in C} x_v - \sum_{v \in N(C)} x_v \leq |C| - 1.$$

More details of the ILP formulations of the 8 concrete problems can be seen in Section B.2 (in the appendix).

Figure 4.12 compares the ILP formulations with the best variants of `Pivot`.

Overall, `Pivot` solves 96% of the instances with small k and 84% of the instances with large k, whereas the ILP solves 85% of the small and 70% of the large instances.

For small k, `Pivot` is significantly faster than the ILP in all three categories. In the degree-constrained category for small k and in the sparse category for all k, `Pivot` solves almost all instances within the time limit. The biggest difference can be observed for the sparse category. This can be somewhat expected since `FixCon` maintains the connectivity constraint during the search, whereas for the ILP, the connectivity constraint is somewhat contrary to the fact that sparse graphs are preferred by the objective functions.

For the dense category and small k, `Pivot` is roughly 20 times faster than the ILP. For the dense category and the degree-constrained category and large k, the ILP and `Pivot` are competitive. A further comparison is shown in Table B.1 (in the appendix);

[2] We also tried using one constraint for each $c \in C$ as Althaus et al. [6]; in preliminary experiments this gave slightly worse results.

the table shows for each considered real-world network the largest k such that all eight CFCO problems could be solved within 600 seconds.

Overall, we conclude that `FixCon` with `Pivot` is competitive with off-the-shelf ILP formulations of the considered CFCO problems when $k \leq 20$ and problem-specific pruning rules are employed.

4.9 Conclusion

We have demonstrated the usefulness of subgraph enumeration for the generic Connected Fixed-Cardinality Optimization problem. For using the generic version of `FixCon` without problem-specific pruning rules, a user only needs to implement the objective function val_G and provide some properties of val_G. We refer to Section B.1 (in the appendix) for the implementations of the eight problems considered in this chapter. Our only assumption is that the objective function val_G depends only on the isomorphism class of $G[S]$. This version can be used as a baseline for comparison with special purpose algorithms or as a base implementation that can be improved by adding problem-specific pruning rules. In the latter setting, `FixCon` is competitive with standard ILP formulations for the problems under consideration, as we showed.

There are many avenues to pursue in future research. First, we could compare our algorithm with further existing approaches. For example, for some objective functions a comparison with `BDDE` [167] is possible. Recall, that in `BDDE` all connected induced subgraphs H that for a given hereditary objective function val_G and a threshold t, fulfill $f(H) > t$ are enumerated. Hence, by setting the threshold to the global upper bound minus one and stopping as soon as one solution is found, one could compare `BDDE` with our algorithm for hereditary graph properties. After these changes a comparison with `FixCon` is directly possible: Currently, there are two example problems which are hereditary, namely Acyclic Subgraph and Triangle-Free Subgraph.

Moreover, further problem-specific rules are highly desirable, particularly for the dense category where our algorithm performs poorly compared with the ILP on sparse graphs. The main reason for this behavior is that the maximal number of edges of any induced subgraph with k vertices in a real-world graph might be significantly lower than the number of edges of a clique of size k. This large gap has the result that our *CJR* is only successful at nodes with a high depth.

Another candidate for a comparison is a mixed integer linear program for the problem of finding a connected induced subgraph with a bound on the maximum degree to maximize the weight of the edges within the subgraph [163]. This mixed

integer linear program could be a good benchmark for CFCO problems since it only used a polynomial number of constraints.

Furthermore, we aim to further improve the generic part of `FixCon`, for example by extending the *Twin Rule* which exploits symmetry in the neighborhood to cases where vertices have almost the same neighborhood. One possible extension is the following: Suppose the graph G contains two vertices u and w such that $N[u] = N[w] \cup \{v\}$. Let C be the connected set of a node in the search tree. If the algorithm already considered all possible vertex sets containing $C \cup \{w\}$ and now considers the connected set $C \cup \{u\}$, then we can immediately also put vertex v into the connected set $C \cup \{u\}$ since otherwise we could replace u with w and hence no better solution can be found.

Another possible line for further speed-ups is the use of color coding [5]. A first option is to use $k + t$ colors to randomly color the vertices of the input graph G. Here t is a non-negative integer serving as a parameter. Then, one searches for a connected induced subgraph $G[S]$ which is colorful, that is, all vertices in S have a different color. This approach sparsifies the graph: We may remove all edges such that both endpoints have the same color. This approach has shown to be useful for the problem to detect paths of length k [5]. A second option is to use only two colors "black" and "white" and search for a solution which is white. Hence, we may remove all black vertices and the use our enumeration algorithms. This approach is closely related to the random separation method [39]: In random separation one additionally requires that the neighborhood of the solution is black. The advantage of our approach is that the success probability is 2^{-k} compared to $2^{-k\Delta}$ of random separation.

In our current implementation we apply each pruning rule in every search tree node. This might not be the fastest way to apply these rules. For example, it could be faster to apply the *VAR* only in those search tree nodes in which the size of the connected set is even. More generally, one could apply a branching rule only in each ith search tree node. It could also be faster to apply some rule only for search tree nodes which corresponding set has size at least $k - c$ where c is a fixed constant. One may define a hyperparameter for each of these options and subsequently optimize these hyperparameters to obtain a small running time with hyperparamter tools like `SMAC` [116].

Currently, our algorithm is implemented in Python. Since Python programs cannot compete with similar programs written in compiled languages in terms of running time, we currently work on an implementation of our algorithm in Java. From this change of programming language alone, we expect a substantial speed-up. Another interesting line of research is to examine how parallelization can be used to

speed up `FixCon`, as it was done for example for clique enumeration [51].

Also, we aim to collect further example problems. It is interesting to study the performance of our algorithm with problems which do not fit into the existing three categories. One example could be: Does G contain a vertex set S of size k such that the induced subgraph $G[S]$ contains no cycle of even length? Is our algorithm also competitive with ILPs for such problems?

Furthermore, we aim to extend our algorithms to allow richer graph models. For example, one could allow vertex and edge weights. In this case, for example, the *Twin Rule* might not work anymore since two twin vertices might have different vertex weights. Another way to extend the graph model is to allow for vertex and edge colors, or directed edges. One could also aim to address problems where the objective function depends not only on the subgraph itself, as we assumed in this chapter, but also on the rest of the graph G. This could be used to solve, for example, Multi-Node Hub [205]. In Multi-Node Hub one aims to find a $(k, n-k)$-cut such that the number of edges in the cut is maximized with the additional constraint that the partite set of size k is connected. Note that this problem is the variant of Max $(k, n-k)$-Cut to which a connectivity constraint is added to the partite set of size k; note that we study the parameterized complexity of Max $(k, n-k)$-Cut in Chapter 8. Other examples include finding a connected dominating set [125], or a connected vertex cover [140].

Another possibility is to extend `FixCon` to enumeration problems, where one wants to output all optimal solutions or to counting problems, where one wants to output their number.

Finally, it is also open to translate other generic subgraph problem representations into `FixCon` objective functions. For example, one could aim to automatically translate first-order-logic formulas [212] into `FixCon` objective functions and, in particular, extract useful properties such as edge-monotonicity or vertex-based upper bounds directly from the formulas.

Chapter 5

Clique Relaxations in (Weakly) Closed Graphs

Clique relaxations have been studied extensively with respect to their parameterized complexity. For a given clique relaxation Π there exist two main tasks: First, find a vertex set of size k that satisfies property Π, and second, enumerate all vertex sets which correspond to maximal elements of Π in the input graph. The most strict model in the context of clique-relaxations are cliques (see Definition 1.1). Clique is the corresponding decision problem of detecting a clique of size at least k. By Clique Enumeration we denote the problem of enumerating all maximal cliques. The most famous algorithm for Clique Enumeration is the Bron-Kerbosch algorithm [34]. An implementation of Bron-Kerbosch by Tomita et al. has proven to be the faster by orders of magnitude in practice than other algorithms for this task [216]. Furthermore, the algorithm of Tomita et al. has a overall running time of $\mathcal{O}(3^{n/3})$ excluding the time to write the output. This running time is optimal: a graph may have up to $3^{n/3}$ maximal cliques [172]. Clique Enumeration has application in the detection of communities in social networks [44, 45] and bioinformatics [233].

As discussed in Chapter 1, in many applications the clique model is often too strict. This resulted in the introduction of clique relaxations. In these models, at least one clique-defining property of being is relaxed [168, 191, 193]. One popular model are s-plexes (see Definition 1.2). s-Plex is the corresponding decision problem. Furthermore, by s-Plex Enumeration we denote the problem of enumerating all maximal s-plexes. There exist fast implementations for s-Plex Enumeration [49, 50]. Another popular model are s-defective cliques (see Definition 1.3). s-Defective Clique is the corresponding decision problem. By s-Defective Clique Enumeration we denote the corresponding problem of enumerating all

maximal s-defective cliques. Yet another popular model are s-clubs (see Definition 1.4). s-Club is the corresponding decision problem. By s-Club Enumeration we denote the corresponding problem of enumerating all maximal s-clubs.

Another popular model for communities are bicliques. In such models the set S of vertices forming this group is partitioned into two sets A and B such that each vertex in A is adjacent to each vertex in B. Often, it is also required that A and B are independent sets. In this case such groups are referred to as *induced bicliques* and otherwise as *non-induced bicliques*. The corresponding decision problem of finding a (non-) induced biclique of size at least k is called (Non-)Induced (k_1, k_2)-Biclique, and the corresponding problem of enumerating all maximal (non-) induced bicliques is called (Non-)Induced Biclique Enumeration.

The decision variants of the above-mentioned models have been studied extensively with respect to the standard parameter solution size k: Clique is W[1]-hard with respect to k [53, 63] and also s-Plex is W[1]-hard when parameterized by k for all $s \in \mathbb{N}$ [131, 144]. Furthermore, one can show that s-Defective Clique is W[1]-hard with respect to k by adapting a previous hardness proof for Densest-k-Subgraph [198, Theorem 20]. Also, Non-Induced (k_1, k_2)-Biclique is W[1]-hard with respect to k_1 even if $k_1 = k_2$ [157] and Induced (k_1, k_2)-Biclique is W[1]-hard even if $k_1 = k_2$ [53]. In contrast, s-Club parameterized by k admits an FPT-algorithm [43, 206]. Furthermore, s-Club does not admit a polynomial kernel for k unless coNP $\subseteq$ NP/poly [206].

Another very important parameter is the maximum degree Δ. Clique admits a trivial FPT-algorithm for Δ since a clique is contained in the closed neighborhood of some vertex. Furthermore, the observation that each s-plex has size at most $\Delta + s$ can be used together with a case distinction whether the s-plex is connected or not to show that s-Plex admits an FPT-algorithm for $\Delta + s$ [143]. This algorithm can be adapted to also solve s-Defective Clique in FPT-time for the parameter $\Delta + s$. A similar observation can be used to show that s-Club is FPT with respect to $\Delta + s$: Each s-club is contained in the sth neighborhood of some vertex. Thus, the size of an s-club is bounded by a function depending on s and Δ.

Since the degeneracy d is a smaller parameter than the maximum degree Δ one important aim in parameterized complexity is to lift positive algorithmic results from Δ to d. Eppstein et al. [70] showed that Clique Enumeration can be solved in $\mathcal{O}(dn3^{d/3})$ time. Furthermore, each graph has at most $(n-d)3^{d/3}$ maximal cliques [70]. The algorithm of Eppstein et al. [70] also implies that Clique admits an FPT-algorithm with respect to d. A simple algorithm can enumerate all maximal s-plexes in $2^d n^{s+\mathcal{O}(1)}$ time [143]. This algorithm can be adapted so enumerate all maximal s-defective cliques in $2^d n^{s+\mathcal{O}(1)}$ time. In contrast, 2-Club is NP-hard

on 6-degenerate graphs [110]. Moreover, Induced Biclique Enumeration can be solved in $\mathcal{O}^*(3^{(\Delta+d)/3})$ time [111]. On the negative side, it is impossible to enumerate all maximal induced bicliques in time $\mathcal{O}^*(f(d))$ for any function f because a graph may have too many maximal induced bicliques [111]: Consider the graph with a single universal vertex u and $(n-1)/3$ disjoint triangles. This graph is 3-degenerate and has $3^{(n-1)/3}$ maximal induced bicliques where one part consists of u.

In recent work, Fox et al. [84] proposed exploiting a different property of real-world graphs that is motivated by the triadic closure principle. This principle postulates that people in a social network which have many common friends are likely to be friends themselves. Many real-world social networks give evidence for this postulate as they contain no pair of non-adjacent vertices with many common neighbors. The degree to which a given graph adheres to the triadic closure principle can be expressed in the *closure number of* G, defined as follows.

Definition 5.1 ([84])**.** Let $\mathrm{cl}_G(v) \coloneqq \max_{v' \in V \setminus N[v]} |N(v) \cap N(v')|$ denote the *closure number* of a vertex v in a graph G. A graph G is *c-closed* if $\mathrm{cl}_G(v) < c$ for all $v \in V(G)$. The *closure number of a graph* G is the smallest integer c such that G is c-closed.

Fox et al. [84] suggested a further graph parameter which combines sparseness and triadic closure, the *weak closure* of a graph.

Definition 5.2 ([84])**.** A graph G is weakly γ-closed[1] if one of the following holds:

- There exists a *weak closure ordering* $\sigma \coloneqq (v_1, \ldots, v_n)$ of G, that is, an ordering such that $\mathrm{cl}_{G_i}(v_i) < \gamma$ for all $i \in [n]$ where $G_i \coloneqq G[\{v_i, \ldots, v_n\}]$.
- Every induced subgraph G' of G has a vertex $v \in V(G')$ such that $\mathrm{cl}_{G'}(v) < \gamma$.

The *weak closure number* of a graph G is the smallest integer γ such that G is weakly γ-closed.

The four parameters maximum degree Δ, degeneracy d, closure c, and the weak closure γ are related as follows:

1. The maximum degree Δ is an upper bound for the d, c, and γ.
2. Since each degeneracy ordering is also a weak closure ordering, we observe that γ is at most $d+1$.

[1] To avoid confusion with the closure number c, we denote the weak closure by γ instead of c.

3. Since the closure of a vertex v in a subgraph of G is bounded from above by the closure of v in G, we observe that γ is at most c.

4. The degeneracy d and the closure number c are incomparable as witnessed by large complete graphs (they have large degeneracy and are 1-closed) and large complete bipartite graphs where one part has size two (they are 2-degenerate and have large closure number).

The latter examples also show that γ can be much smaller than the closure number c and the degeneracy d. To conclude: d and c are independent and smaller than Δ, and γ is a smaller parameter than d and c. Consequently, FPT-algorithms for d and c are, in principle, preferable to those for the maximum degree Δ. Similar, FPT-algorithms for γ are, in principle, preferable to those for the closure number c or the degeneracy d. From an application point of view, the weak closure number is also an excellent parameter since it tends to take on very small values in real-world networks [84] (see also Table 5.1).

Fox et al. [84] showed that a graph has $\mathcal{O}(3^{\gamma/3} \cdot n^2)$ many maximal cliques which, using known clique enumeration algorithms, gives an algorithm that enumerates all maximal cliques in $\mathcal{O}^*(3^{\gamma/3})$ time. Independently, Behera et al. [17] obtained similar results for the enumeration of maximal s-plexes and further dense subgraphs parameterized by the c-closure; it seems that their algorithms for s-plex enumeration can be adapted to parameterization by weak closure as well [17].

The parameters closure c and the weak closure γ are not only helpful to obtain fast (FPT-)algorithms for clique relaxations. Koana et al. showed that Independent Set has a kernel with at most ck^2 vertices [139]. Later, Koana et al. improved this result by showing that Independent Set has a kernel with at most γk^2 vertices [137]. Koana et al. also showed that each c-closed graph with at least $(c-1)\binom{b-1}{2} + (a-1)(b-1)$ vertices contains a clique of size a or an independent set of size b [139]. Also, it was shown that many domination problems admit FPT-algorithms with respect to the (weak) closure. Koana et al. provided an almost tight kernel of size $k^{\mathcal{O}(c)}$ for Dominating Set. Recently, Lokshtanov and Surianarayanan showed that Dominating Set parameterized by $\gamma + k$ can be solved in $\mathcal{O}^*(k^{\mathcal{O}(\gamma^2 k^3)})$ time [160]. Furthermore, Kanesh et al. [125] showed that Perfect Code and Connected Dominating Set admit an FPT-algorithm for the parameter $k + c$. Also, we provided kernels of size $k^{\mathcal{O}(\gamma)}$ for Connected Vertex Cover and Capacitated Vertex Cover. Furthermore, in Chapter 8 we provide almost tight kernels of size $k^{\mathcal{O}(c)}$ for Maximum Partial Vertex Cover and Max $(k, n-k)$-Cut. Finally, Koana and Nichterlein [142] provided parameterized polynomial-time algorithms for detecting and enumerating small graphs (3 or 4

Table 5.1: A comparison of the number n of vertices, number m of edges, the maximum degree Δ, the closure c, the degeneracy d and the weak closure γ in social and biological networks.

Instance name	n	m	Δ	c	d	γ
adjnoun-adjacency	112	425	49	14	6	6
arenas-jazz	198	2 742	100	42	29	18
ca-netscience	379	914	34	5	8	3
bio-celegans	453	2 025	237	26	10	9
bio-diseasome	516	1 188	50	9	10	5
soc-wiki-Vote	889	2 914	102	18	9	8
arenas-email	1 133	5 451	71	19	11	8
bio-yeast	1 458	1 948	56	8	5	4
ca-CSphd	1 882	1 740	46	3	2	3
soc-hamsterster	2 426	16 630	273	77	24	19
ca-GrQc	4 158	13 422	81	43	43	9
soc-advogato	5 167	39 432	807	218	25	21
bio-dmela	7 393	25 569	190	72	11	12
ca-HepPh	11 204	117 619	491	90	238	54
ca-AstroPh	17 903	196 972	504	61	56	30
soc-brightkite	56 739	212 945	1 134	184	52	49

vertices) in c-closed graphs.

Our Results. In a nutshell, we show that low (weak) closure helps in detecting or enumerating s-plexes, s-defective cliques, s-clubs, and (non-)induced bicliques. Our main results are listed in Table 5.2.

Our results improve over the state of the art in the following sense: the best known tractability results for these problems employ the degeneracy of the input graph as a parameter and, as discussed above, the weak closure is essentially a smaller parameter. For some problems, we also provide results for the c-closure parameter. There are two reasons for this. First, for some problems we obtain better running time bounds for the parameter c. Second, we provide some lower bounds for the problems under consideration and, whenever possible, we provide them for the larger closure parameter c.

From a practical point of view, the most important results are, in our opinion, the enumeration algorithms for maximal non-induced bicliques and maximal s-plexes

Table 5.2: An overview of our results. Our algorithms for s-Plex Enumeration and s-Defective Clique Enumeration can be used directly to solve the correspondence decision problems s-Plex (Corollary 5.4) and s-Defective Clique (Corollary 5.7) in the same running times. Furthermore, our algorithm for Non-Induced-Biclique Enumeration can be adapted to solve also Non-Induced (k_1, k_2)-Biclique in $\mathcal{O}^*(2^\gamma)$ time (Theorem 5.15).

Problem	Result	Reference
s-Plex Enumeration	$\mathcal{O}(2^\gamma n^{2s+1})$-time algorithm for $s \geq 2$	Theorem 5.3
s-Plex	W[1]-hard for k even if $c = 2$	Theorem 5.5
s-Defective Clique Enumeration	$\mathcal{O}(2^\gamma n^{s+3})$-time algorithm	Theorem 5.6
s-Defective Clique	W[1]-hard for k even if $c = 2$	[198]
	$2^{\mathcal{O}(\gamma\sqrt{s}+s\log k)} n^{\mathcal{O}(\sqrt{s})}$-time algorithm	Theorem 5.10
2-Club	NP-hard for $c = 4$	Theorem 5.13
Non-Induced-Biclique Enumeration	$\mathcal{O}^*(2^\gamma)$-time algorithm	Theorem 5.14
Induced (k, k)-Biclique	$\mathcal{O}^*(\gamma^{\mathcal{O}(\gamma)})$-time algorithm	Theorem 5.17
Induced (k_1, k_2)-Biclique	$\mathcal{O}^*(1.6107^c)$-time algorithm if $k_1 \geq 2$	Theorem 5.18
	NP-hard if $k_1 = 1$ for $c = 3$ and $\gamma = 2$	Theorem 5.20
	P for $c = 2$	Corollary 5.23
	P for $k_1 = 1$ and $\gamma = 1$	Theorem 5.21
	P for $k_1 \geq 2$ and $\gamma \leq k_1 + 1$	Theorem 5.21
	NP-hard for $k_1 \geq 2$ and $\gamma \geq k_1 + 2$	Theorem 5.21

whose running times grow moderately with γ. Both algorithms are based on the algorithm to enumerate all maximal cliques in weakly γ-closed graphs [84].

5.1 Clique Relaxations

In this section, we present algorithms for relaxations of the Clique problem. For the positive results, we only consider parameterization by the weak closure number γ.

5.1.1 s-Plex

First, we study the problem of enumerating all maximal s-plexes.

Theorem 5.3. *For $s \geq 2$, a graph G has $\mathcal{O}(2^\gamma n^{2s-1})$ maximal s-plexes. Moreover, all maximal s-plexes of G can be enumerated in $\mathcal{O}(2^\gamma n^{2s+1})$ time.*

Proof. First, we show the bound on the number of maximal s-plexes in a weakly γ-closed graph. Let $v \in V(G)$ be a vertex such that $\mathrm{cl}_G(v) < \gamma$ and let $G' \coloneqq G - v$ be the graph obtained by deleting v. Let $\mathcal{S}$ and $\mathcal{S}'$ be the collections of all maximal s-plexes (without duplicates) in G and G', respectively. We show that $|\mathcal{S}| \leq |\mathcal{S}'| + 2^\gamma n^{2s-2}$ and that $\mathcal{S}$ can be constructed from $\mathcal{S}'$ in $\mathcal{O}(|\mathcal{S}'| \cdot n + 2^\gamma n^{2s+1})$ time. To obtain the bound we identify the following four types of maximal s-plexes in G:

Type 1: S does not contain v. Then, S is also maximal in G'.

Type 2: S contains v and $S \setminus \{v\}$ is maximal in G'.

Type 3: S contains v, $S \setminus \{v\}$ is not maximal in G', and S contains a non-neighbor of v (that is, $S \setminus N_G(v) \neq \emptyset$).

Type 4: S contains v, $S \setminus \{v\}$ is not maximal in G', and S is contained in the neighborhood of v, that is, $S \subseteq N_G[v]$.

Clearly, each maximal s-plex is of one of these four types. It is easy to see that there are $|\mathcal{S}'|$ maximal s-plexes of Type 1 and Type 2. Hence, it remains to bound the number of maximal s-plexes of Type 3 and Type 4.

Next, we bound the number of maximal s-plexes of Type 3. Consider such an s-plex S. We may partition S into three parts as follows: We first divide S into $S_v \coloneqq S \cap N_G[v]$ and $\widetilde{S_v} \coloneqq S \setminus N_G[v]$. We divide S_v further into $S_{uv} \coloneqq S_v \cap N_G(u)$ and $\widetilde{S_{uv}} \coloneqq S_v \setminus N_G(u)$ for some vertex $u \in \widetilde{S_v}$. Here, u is any non-neighbor of v to exploit the weak γ-closure. By the definition of s-plexes, $|\widetilde{S_v}| < s$ and $|\widetilde{S_{uv}}| < s$. Hence, there are at most n^{2s-2} choices for $\widetilde{S_v}$ and $\widetilde{S_{uv}}$. For S_{uv}, there are at most $2^{\gamma-1}$ choices because $S_{uv} \subseteq N_G(v) \cap N_G(u)$ and $|N_G(v) \cap N_G(u)| < \mathrm{cl}_G(v) < \gamma$. Overall, there are at most $2^{\gamma-1} n^{2s-2}$ maximal s-plexes of Type 3.

It remains to bound the number of maximal s-plexes of Type 4. Let S be one of these s-plexes. Since $S' \coloneqq S \setminus \{v\}$ is not maximal in G', there exists a vertex $u \in V(G) \setminus S$ such that $S' \cup \{u\}$ is an s-plex in G'. If $u \in N_G(v)$, then $S \cup \{u\}$ is also an s-plex in G, which contradicts the fact that S is maximal in G. Hence, we can assume that $u \notin N_G(v)$. Then, $S \setminus N_G(u)$ contains at most $s - 1$ vertices, which in turn implies that there are at most n^{s-1} choices for $S \setminus N_G(u)$. Since $S \subseteq N(v)$ we observe that $S \cap N_G(u) \subseteq N_G(v) \cap N_G(u)$ and $|N_G(v) \cap N_G(u)| \leq \mathrm{cl}_G(v) < \gamma$. Thus, we have $2^{\gamma-1}$ choices for $S \cap N_G(v)$. All in all, there are at most $2^{\gamma-1} n^s$ maximal s-plexes of Type 4.

By the above analysis, we obtain $|\mathcal{S}| \leq |\mathcal{S}'| + 2^{\gamma-1}n^{2s-2} + 2^{\gamma-1}n^s \leq |\mathcal{S}'| + 2^{\gamma}n^{2s-2}$. Next, we bound the overall number of maximal s-plexes in a graph with n vertices. To this end, let a_n be the number of maximal s-plexes in weakly γ-closed graphs on n vertices. Clearly, $a_1 = 1$. Furthermore, the above analysis showed that $a_n - a_{n-1} = |\mathcal{S}| - |\mathcal{S}'| \leq 2^{\gamma}n^{2s-2}$. Hence, by induction we obtain $a_n = a_1 + \sum_{i=2}^{n}(a_i - a_{i-1}) \leq 2^{\gamma}n^{2s-1} + 1$. In other words, a weakly γ-closed graph on n vertices has at most $2^{\gamma}n^{2s-1} + 1$ maximal s-plexes.

Second, we bound the overall time needed to enumerate all maximal s-plexes. To obtain this bound, we again let $v \in V(G)$ be any vertex such that $\mathrm{cl}_G(v) < \gamma$, let $G' \coloneqq G - v$ be the graph obtained by deleting v, and let $\mathcal{S}$ and $\mathcal{S}'$ be the collections of all (without duplicates) maximal s-plexes in G and G'. Observe that all maximal s-plexes of Type 1 and 2 can be found in $\mathcal{O}(|\mathcal{S}'| \cdot n)$ time. Furthermore, maximal s-plexes of Type 3 and 4 can be enumerated in $\mathcal{O}((2^{\gamma-1}n^{2s-2} + 2^{\gamma-1}n^s) \cdot n^2)$ time, because it takes $\mathcal{O}(n^2)$ time to verify whether a vertex set is a maximal s-plex or not. Finally, we remove duplicates in $\mathcal{O}((|\mathcal{S}'| + 2^{\gamma-1}n^{2s-2} + 2^{\gamma-1}n^s) \cdot n) = \mathcal{O}(|\mathcal{S}'| \cdot n + 2^{\gamma}n^{2s-1})$ time, using radix sort. Altogether, the algorithm needs $\mathcal{O}(|\mathcal{S}'| \cdot n + 2^{\gamma}n^{2s})$ time to enumerate all maximal s-plexes in G. Recall that a_n is the number of maximal s-plexes in a weakly γ-closed graph on n vertices. Thus, all maximal s-plexes of a weakly γ-closed graph on n vertices can be enumerated in $\mathcal{O}((a_n \cdot n + 2^{\gamma}n^{2s}) \cdot n) = \mathcal{O}(2^{\gamma}n^{2s+1})$ time. □

A factor of n^{2s-2} for the number of maximal s-plexes in Theorem 5.3 is unavoidable: Consider a graph G consisting of two cliques C_1 and C_2 of equal size. Clearly, G is 1-closed. Each subset of C_1 of size exactly $s-1$ and each subset of C_2 of size exactly $s-1$ together form a maximal s-plex. Hence, there exist 1-closed graphs with $\Omega((n/2)^{2s-2})$ maximal s-plexes.

For s-Plex, Theorem 5.3 directly implies the following.

Corollary 5.4. *For $s \geq 2$, s-Plex can be solved in $\mathcal{O}(2^{\gamma}n^{2s+1})$ time.*

Next, we show that there is presumably no $f(k) \cdot n^{\mathcal{O}(1)}$-time algorithm for s-Plex in 2-closed graphs. Moreover, our reduction also shows that s-Plex is W[1]-hard for the parameter $k + s + d$.

Theorem 5.5. *s-Plex is W[1]-hard in 2-closed graphs when parameterized by $k + s + d$.*

Proof. We reduce from Clique. An illustration of our construction is shown in Figure 5.1. Let (G, k) be an instance of Clique with $k \geq 4$. First, we subdivide each edge uv of G twice. That is, we remove the edge uv and add edges ux_u^v, $x_u^v x_v^u$, and $x_v^u v$, where x_u^v and x_v^u are two new vertices. Second, for each edge $uv \in E(G)$,

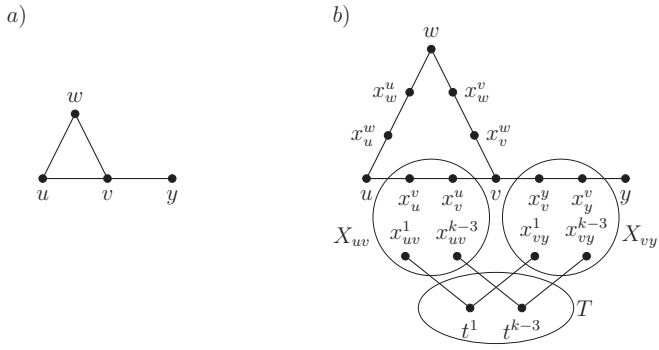

Figure 5.1: Illustration of the construction of Theorem 5.5. *a*) shows the graph G of the Clique instance and *b*) shows the graph G' of s-Plex. Here, the sets X_{uw} and X_{vw} are not drawn. Note that the sets X_{uv} and X_{vy} are cliques containing exactly $k-1$ vertices.

we introduce $k-3$ vertices $x_{uv}^1, \ldots, x_{uv}^{k-3}$. Let $X_{uv} \coloneqq \{x_u^v, x_v^u, x_{uv}^1, \ldots, x_{uv}^{k-3}\}$ and let $X \coloneqq \bigcup_{uv \in E(G)} X_{uv}$. We then add edges so that X_{uv} forms a clique. Lastly, we introduce a set $T \coloneqq \{t^1, \ldots, t^{k-3}\}$ of $k-3$ vertices and add edges between x_{uv}^i and t^i for each $uv \in E(G)$ and each $i \in [k-3]$. Let G' be the resulting graph.

It is easy to verify that G' is 2-closed. Moreover, G' is $(k-1)$-degenerate: Each vertex $x \in X_{uv}$ is of degree $k-1$ and there is no edge in $G'-X$. We show that G has a clique of size k if and only if G' has an s-plex of size k', where $k' \coloneqq 2k-3+(k-1)\binom{k}{2}$ and $s \coloneqq k' - (k-1)$.

Suppose that G has a clique S of size exactly k. Let $S' = S \cup T \cup \bigcup_{u,v \in S} X_{uv}$. Observe that $|S'| = k'$. We verify that each vertex in $G'[S']$ has degree at least $k' - s = k - 1$.

- Let $v \in S$. By construction, we have $x_v^u \in N_{G'}(v)$ for each $u \in S \setminus \{v\}$. Since x_v^u is contained in S', v has at least $k-1$ neighbors in $G'[S']$.
- We have $\deg_{G'[S']}(t^i) \geq \binom{k}{2} \geq k-1$ for each $i \in [k-3]$, because t^i is adjacent to x_{uv}^i for all $uv \in E(G[S])$.
- Consider x_u^v for $uv \in E(G[S])$. We have $u \in N_{G'}(x_u^v)$ by construction. Moreover, x_u^v is adjacent to all $k-2$ vertices in $X_{uv} \setminus \{x_u^v\}$. Thus, we have $\deg_{G'[S']}(x_u^v) \geq k-1$.
- Consider x_{uv}^i for $uv \in E(G[S])$ and $i \in [k-3]$. We have $t^i \in N_{G'}(x_{uv}^i)$ by

construction. Moreover, x^i_{uv} is adjacent to all $k-2$ vertices in $X_{uv} \setminus \{x^i_{uv}\}$. Thus, we have $\deg_{G'[S']}(x^i_{uv}) \geq k-1$.

Thus, every vertex has at least $k-1 = k'-s$ neighbors in $G'[S']$.

Conversely, suppose that S' is an s-plex of size exactly k'. We start with the following claim.

Claim 2. If S' contains a vertex x of X_{uv} for some $uv \in E(G)$, then S' also contains all vertices in $N_{G'}[X_{uv}]$, that is, $\{u,v\} \cup X_{uv} \cup T \subseteq S'$.

Proof of Claim. By construction, $\deg_{G'}(x) = k-1$. Since each vertex in $G'[S']$ has degree $|S'|-s \geq k-1$ by the definition of s-plexes, we have $N_{G'}[X_{uv}] \subseteq S'$. ■

Let $\ell = |S' \cap V(G)|$. We conclude that there are at most $\binom{\ell}{2}$ edges $uv \in E(G)$ with $X_{uv} \cap S' \neq \emptyset$ since otherwise the above claim would imply that $|S' \cap V(G)| > \ell$. By construction, we have $|X_{uv}| = k-1$ for each $uv \in E(G)$. Thus, we have

$$|S'| = |S' \cap V(G)| + |T| + |S' \cap X| \leq \ell + k - 3 + (k-1)\binom{\ell}{2}.$$

Since $|S'| = k' = 2k-3+\binom{k}{2}$, we obtain $\ell \geq k$.

By definition, each vertex $v \in S' \cap V(G)$ has at least $|S'|-s \geq k-1$ neighbors in $G'[S']$. So there are at least $\ell(k-1)/2$ edges $uv \in E(G)$ such that $S' \cap X_{uv} \neq \emptyset$. From the above claim we know that $X_{uv} \subseteq S'$ for each X_{uv} with $X_{uv} \cap S' \neq \emptyset$. Hence, we obtain that

$$|S'| \geq |S' \cap V(G)| + |T| + |S' \cap X| \geq \ell + k - 3 + (k-1) \cdot \ell(k-1)/2.$$

Since $|S'| = k' = 2k-3+(k-1)\binom{k}{2}$, we obtain $\ell = k$ and $|S' \cap X| = (k-1)\binom{k}{2}$. Since $|S' \cap X| = (k-1)\binom{k}{2}$ we conclude that each two vertices in $S' \cap V(G)$ are adjacent. Thus, $S' \cap V(G)$ is a clique of k vertices in G by construction. □

5.1.2 s-Defective Clique

First, we study the problem of enumerating *all* maximal s-defective cliques in weakly γ-closed graphs. To this end, we adapt the algorithm of Theorem 5.3 to obtain an algorithm to enumerate all maximal s-defective cliques.

The only difference to the proof of Theorem 5.3 is the following: For bounding the number of s-plexes of Type 3 the sets $\widetilde{S_v}$ and $\widetilde{S_{uv}}$ were bounded by $s-1$ each. Since a maximal s-defective clique contains at most s non-edges and $uv \notin E(G)$ we observe that $|\widetilde{S_v} \cup \widetilde{S_{uv}}| < s$. Hence, there are at most $2^{\gamma-1}n^{s-1}$ maximal s-defective

cliques of Type 3. Thus, we can bound the overall number of maximal s-defective cliques by $2^\gamma n^{s+1} + 1$. Since the rest of the proof is completely analogous, we omit it.

Theorem 5.6. *For $s \geq 2$, there are $\mathcal{O}(2^\gamma n^{s+1})$ maximal s-defective cliques in weakly γ-closed graphs and they can be enumerated in $\mathcal{O}(2^\gamma n^{s+3})$ time.*

A factor of n^{s+1} in the number of maximal s-defective cliques in Theorem 5.6 is inevitable due to the following lower bound: Again we consider the graph G consisting of two disjoint cliques C_1 and C_2, each of size $n/2$. For each clique $C \subseteq C_1$ of size s and each $v \in C_2$, the vertex set $C \cup \{v\}$ is a maximal s-defective clique. Thus, G has $\Omega((n/2)^{s+1})$ maximal s-defective cliques.

Second, we study s-Defective Clique, the decision problem of finding a sufficiently large s-defective clique. Theorem 5.6 directly implies the following.

Corollary 5.7. *s-Defective Clique can be solved in $\mathcal{O}(2^\gamma n^{s+3})$ time.*

Next, we present faster algorithms in terms of the dependence on s. First, we show that each s-defective clique can be covered by $\mathcal{O}(\sqrt{s})$ maximal cliques.

Lemma 5.8. *Let S be an s-defective clique for $s \geq 1$. Then, there is a collection $\mathcal{C}$ of at most $\mathcal{O}(\sqrt{s})$ cliques such that $S \subseteq \bigcup_{C \in \mathcal{C}} C$.*

Proof. Let H denote the complement graph of $G[S]$. By definition, H has at most s edges. Since a clique becomes an independent set in the complement graph, it suffices to show that there is an $\mathcal{O}(\sqrt{s})$-coloring of H (that is, $\chi(H) = \mathcal{O}(\sqrt{s})$). Although this is known folklore, we describe its proof for the sake of completeness. Consider an optimal coloring. Then, for each pair of colors, say red and blue, there is at least one edge with one endpoint red and the other blue (otherwise we find a coloring with fewer colors). Recall that H is the complement graph of $G[S]$. Hence, H has at most s edges, we obtain $s \geq \binom{\chi(H)}{2}$, or equivalently, $\chi(H) \leq \sqrt{2s + \frac{1}{4}} + \frac{1}{2}$. □

Note that a trivial brute-force algorithm can enumerate all (not necessarily maximal) cliques in $\mathcal{O}(2^d dn)$ time. Lemma 5.8 says that each s-defective clique is covered by at most $\mathcal{O}(\sqrt{s})$ cliques. Hence, by a simple brute-force we obtain the following.

Theorem 5.9. *s-Defective Clique can be solved in $2^{\mathcal{O}(d\sqrt{s})} n^{\mathcal{O}(\sqrt{s})}$ time.*

We can also use Lemma 5.8 to obtain an algorithm in terms of the smaller parameter γ instead of the degeneracy d without increasing the exponent of n.

Theorem 5.10. *s-Defective Clique can be solved in $2^{\mathcal{O}(\gamma\sqrt{s} + s \log k)} n^{\mathcal{O}(\sqrt{s})}$ time.*

Proof. We first enumerate all maximal cliques in $(3^{\gamma/3} \cdot n^{\mathcal{O}(1)})$ time [84]. If there is a clique of size at least k, then return Yes, since each clique is also an s-defective clique. Now, we assume that there is no clique of size at least k. By Lemma 5.8, it suffices to check whether there is an s-defective clique of size k in $\bigcup_{C \in \mathcal{C}} C$ for each collection $\mathcal{C}$ of $\mathcal{O}(\sqrt{s})$ maximal cliques. Observe that each fixed collection in $\mathcal{C}$ has $\mathcal{O}(k\sqrt{s})$ vertices. Let $W_{\mathcal{C}}$ denote the vertex set of $\mathcal{C}$. By applying the algorithm of Corollary 5.7 to find the largest s-defective clique, we can determine in $\mathcal{O}(2^{\gamma}(\sqrt{s}k)^{\mathcal{O}(s+3)})$ time whether $W_{\mathcal{C}}$ contains an s-defective clique of size at least k. Since there are $\mathcal{O}^*(3^{\gamma/3})$ maximal cliques, the overall running time of this algorithm is $(3^{\gamma/3} \cdot n^{\mathcal{O}(1)})^{\mathcal{O}(\sqrt{s})} \cdot \mathcal{O}(2^{\gamma}(\sqrt{s}k)^{\mathcal{O}(s+3)}) = 2^{\mathcal{O}(\gamma\sqrt{s}+s\log k)} n^{\mathcal{O}(\sqrt{s})}$ time. □

For c-closed graphs, we can obtain an algorithm whose running time does not depend on k. This is due to the following lemma.

Lemma 5.11. *Let $S \subseteq V(G)$ be an s-defective clique in G, in which at least one pair of vertices is nonadjacent. Then, $|S| \leq c + s$.*

Proof. Let $u, v \in S$ be vertices such that $uv \notin E(G)$. We show that $|S'| \leq c + s - 2$ for $S' \coloneqq S \setminus \{u, v\}$. Since G is c-closed, there are at most $c - 1$ vertices in S' adjacent to both u and v. Moreover, there are at most $s - 1$ vertices in S' which are nonadjacent to either u or v in S', by the definition of s-defective cliques. Thus, we obtain $|S'| \leq (c - 1) + (s - 1) = c + s - 2$. □

From Lemma 5.11 we directly obtain the following.

Corollary 5.12. s-Defective Clique *can be solved in* $2^{\mathcal{O}(c\sqrt{s}+s\log(c+s))} n^{\mathcal{O}(\sqrt{s})}$ *time.*

5.1.3 2-Clubs

Recall that 2-Club is FPT for the parameter maximum degree Δ. In contrast, 2-Club is W[1]-hard with respect to h-index and it is NP-hard on 6-degenerate graphs [110]. Since $\gamma \leq d + 1$, this also implies NP-hardness for constant values of γ. We extent these results, by showing that 2-Club remains NP-hard even on 4-closed graphs.

Theorem 5.13. 2-Club *remains NP-hard even on* 4*-closed graphs.*

Proof. We reduce from Clique.

Construction. Let (G, k) be an instance of Clique. We construct an equivalent instance (G', k') of 2-Club such that G' is 4-closed. We set $k' \coloneqq k \cdot n^2$. For each vertex $w \in V(G)$, we add a clique $K_w \coloneqq \{w_j \mid j \in \{0, \ldots, n^2 - 1\}\}$ of size n^2 to G'.

We denote the graph constructed so far by G^0. Furthermore, let $(e_1, e_2, \ldots, e_m)$ be an arbitrary but fixed ordering of the edges in $E(G)$. We will add edges corresponding to each edge $e_i \in E(G)$ to G'. We denote by G^i the graph after we added the gadgets for the edges e_1 to e_i to G^0. Note that G^0 is the graph constructed so far; a disjoint union of cliques, and that $G^m = G'$. The idea for the gadget of edge $e_i = uv$ is as follows: We add a matching between the vertices of the cliques K_u and K_v. More precisely, we add the edges $u_i v_{i+\ell_{uv} \bmod n^2}$ for each $i \in \{0, \ldots, n^2 - 1\}$ and some fixed integer ℓ_{uv}. We call ℓ_{uv} the *shift* of uv. We will assume that $\ell_{uv} + \ell_{vu} = n^2$. To simplify notation, we will assume that the modulo n^2 is taken after the addition of a shift. The difficult part lies in choosing ℓ_{uv} carefully to obtain a graph with constant closure.

For a vertex pair (a, b) of G by A_{ab} we denote the set of vertices in the cliques K_a and K_b and by B_{ab} the remaining vertices of $V(G')$. Next, we prove the following invariant which is an essential ingredient to show that $G' = G^m$ has constant closure number.

> *Invariant.* For each i, there is a shift ℓ_{uv} for the gadget of the ith edge $e_i = uv$ such that for each two nonadjacent vertices $x \in K_a$ and $y \in K_b$ for any vertices $a, b \in V(G)$ we have $|N_{G^i}(x) \cap N_{G^i}(y) \cap B_{ab}| \leq 1$. Moreover, we can find ℓ_{uv} is polynomial time.

That is, we want to maintain the invariant that two nonadjacent vertices in $K_a \cup K_b$ have at most one common neighbor in B_{ab}. Recall that G^0 is a disjoint union of cliques. Thus, the invariant holds for G^0. In the following, we assume that the invariant holds for the graph G^{i-1}. Recall that the graph G^i is constructed from G^{i-1} by adding the matching for the edge $e_i = uv$. We will show that the invariant can be maintained for G^i. More precisely, we show that we can compute a shift $\ell_{uv} \in \{0, \ldots, n^2 - 1\}$ in polynomial time such that adding the edges $u_j v_{j+\ell_{uv}}$ for each $j \in \{0, \ldots, n^2 - 1\}$ to G^{i-1} does not violate the invariant.

Assume to the contrary that the invariant is violated by two nonadjacent vertices in G^i. Observe that there could be three possibilities on how the invariant could be violated in G^i:

Case 1: Two nonadjacent vertices in A_{pq} for $p, q \in V(G) \setminus \{u, v\}$ violate the invariant,

Case 2: two nonadjacent vertices in A_{uv} violate the invariant, or

Case 3: two nonadjacent vertices in A_{wp} for $w \in \{u, v\}$ and $p \in V(G) \setminus \{u, v\}$ violate the invariant.

In the following, we show that we can choose the shift ℓ_{uv} in such a way to fulfill the invariant also for G^i. We distinguish the three above cases:

Case 1. Let x and y be a pair of nonadjacent vertices in A_{pq} violating the invariant. Note that each edge added to G^{i-1} to obtain G^i is of the form $u_r v_s$. Clearly, $u_r, v_s \notin A_{pq}$. Hence, we conclude that $|N_{G^i}(x) \cap N_{G^i}(y) \cap B_{pq}| = |N_{G^{i-1}}(x) \cap N_{G^{i-1}}(y) \cap B_{pq}| \leq 1$ since the invariant holds for G^{i-1}. Thus, this case is not possible.

Case 2. Let x and y be a pair of nonadjacent vertices in A_{uv} violating the invariant. As in case 1, since only edges with both endpoints in A_{uv} are added to the graph G', we obtain that $|N_{G^i}(x) \cap N_{G^i}(y) \cap B_{uv}| = |N_{G^{i-1}}(x) \cap N_{G^{i-1}}(y) \cap B_{uv}| \leq 1$ since the invariant holds for G^{i-1}. Thus, this case is also not possible.

Case 3. Without loss of generality, assume that $w = u$. Recall that adding a matching between the cliques K_u and K_v can increase the number of common neighbors in B_{up} of two nonadjacent vertices in A_{up} by at most 1. Thus, two vertices in A_{up} violating the invariant in G^i have a common neighbor in some clique K_t in G^{i-1}. Since only the matchings corresponding to the edges $ut, pt \in E(G)$ result in edges between K_u and K_t and between K_p and K_t, the matchings corresponding to the edges ut and pt are already added to G^{i-1}. To obtain G^i from G^{i-1} only a matching between K_u and K_v is added. Thus, we conclude that the matching corresponding to the edge pv was already present in G^{i-1}.

For every $j \in \{0, \ldots, n^2 - 1\}$, we have $N(u_j) \cap K_t = \{t_{j+\ell_{ut}}\}$ and $N(t_{j+\ell_{ut}}) \cap K_p = \{p_{j+\ell_{ut}+\ell_{tp}}\}$. Hence, u_j and $p_{j'}$ have a common neighbor in K_p if and only if $j' - j \equiv \ell_{ut} + \ell_{tp}$. Similarly, u_j and $p_{j'}$ have a common neighbor in K_v if and only if $j' - j \equiv \ell_{uv} + \ell_{vp}$. Consequently, the invariant is only violated if $\ell_{uv} \equiv \ell_{ut} + \ell_{tp} + \ell_{pv}$. Thus, for each p and t, there is at most one shift violating the invariant, amounting to at most $(n-2)^2$ forbidden shifts. Since there are n^2 possible shifts, we conclude that we can choose a shift ℓ_{uv} in a way which does not violate the invariant. Note that this does not only show the existence of a shift maintaining the invariant, the above argument also shows that the shift ℓ_{uv} can be constructed in polynomial time, although no explicit formula for ℓ_{uv} is given here.

Thus, we have shown that the invariant is maintained for each i, in particular for $i = m$ and hence for the resulting graph G'.

Bounded Closure. We use the invariant to show that G' is 4-closed. Consider two nonadjacent vertices $x \in K_u$ and $y \in K_v$ in G'. Observe that $u \neq v$ since otherwise $xy \in E(G')$. By the invariant, we have $|N_{G'}(x) \cap N_{G'}(y) \cap B_{uv}| \leq 1$. Recall that $A_{uv} = K_u \cup K_v$. Since x has at most one neighbor in K_v and since y has at most one neighbor in K_u, we conclude that x and y have at most three common neighbors. Thus, G' is 4-closed.

Correctness. Suppose that G contains a clique C of size at least k. Let $S \coloneqq$

$\{K_v \mid v \in C\}$. Clearly, S has size $k' = k \cdot n^2$. It remains to show that S is a 2-club. Consider two nonadjacent vertices $x, y \in S$. Note that $x \in K_u$ and $y \in K_v$ for $u, v \in C$ such that $u \neq v$ since otherwise $xy \in E(G')$. Since C is a clique, we have $uv \in E(G)$ and thus we added a matching between the cliques K_u and K_v. Hence, x has a neighbor z in K_v and thus x and y have distance 2 since K_v is a clique.

Conversely, suppose that S contains an 2-club S of size at least $k' = k \cdot n^2$. Let $T \coloneqq \{v \mid |K_v \cap S| \geq n + 1\}$. Observe that $|T| \geq k$, since otherwise $|S| \leq |T| \cdot n^2 + (n - |T|) \cdot n \leq kn^2 - (k-1)n$. In the following, we show that T is a clique in G. Assume towards a contradiction that T is not a clique and let $u, v \in T$ such that $uv \notin E(G)$. Let u^* be a vertex in $K_u \cap S$ and let $U \coloneqq N(u^*) \setminus K_u$. Note that since $uv \notin E(G)$ we have $U \cap K_v = \emptyset$. Furthermore, note that by construction each vertex $y \in K_w$ has at most one neighbor in K_x for any $w, x \in V(G)$ such that $w \neq x$. Thus, $|U| \leq n$. Furthermore, by the same argument we obtain that each vertex in U has at most 1 neighbor in K_v. Thus, u^* has distance at most 2 to at most n vertices in K_v. This is a contradiction to the fact that $|S \cap K_v| \geq n + 1$ and that S is an 2-club. Hence, T is a clique and thus G contains a clique of size at least k. □

We leave the complexity of 2-Club on 2-closed graphs and 3-closed graphs open. We want to point out that 2-closed graphs of diameter two are also known to be *geodetic*, that is, each pair of vertices has a unique shortest path between them. Moreover, it is known that every 2-closed graph G of diameter two satisfies one of the following [23]:

- G contains a vertex v such that $N(v) = V(G)$, or
- G is strongly regular, that is, G is regular and for some $\lambda, \mu \in \mathbb{N}$, every two adjacent (nonadjacent) vertices have λ (μ, respectively) common neighbors (note that $\mu = 1$ since G is 2-closed), or
- G has exactly two vertex degrees.

To show that 2-Club in 2-closed graph is solvable in polynomial time exploiting these three properties might be helpful.

For 2-clubs we only studied the decision variant 2-Club in which we ask for an sufficiently large 2-club in c-closed graphs. The enumeration of all maximal 2-clubs is *not* possible in FPT-time even for graphs with constant closure: Observe that in the construction of Theorem 5.13, C is a maximal clique in the graph G of the Clique instance if and only if $\{K_c \mid c \in C\}$ is a maximal 2-club in the graph G' of the 2-Club instance. The number of maximal cliques in an n-vertex graph is $3^{n/3}$ [172].

Hence, the above correspondence shows that even a 4-closed graph may have up to $3^{n/3}$ maximal 2-clubs.

5.2 Bicliques

The counterpart of cliques in bipartite graphs are (non-) induced bicliques. In this section we study the parameterized complexity of enumerating all maximal (non-) induced bicliques and finding a sufficiently large (non-) induced biclique in (weakly) closed graphs.

5.2.1 Non-Induced Biclique

In this subsection, we study problems of finding non-induced maximal bicliques fulfilling certain cardinality constraints. We also consider Non-Induced Max-Edge Biclique where we demand that $|S| \cdot |T| \geq k$ instead of putting constraints on the partition sizes. Non-Induced Max-Edge Biclique can be solved by solving $\sqrt{k}$ instances of Non-Induced (k_1, k_2)-Biclique and thus the latter problem can be considered to be more difficult in our setting. Non-Induced Max-Edge Biclique can be solved in $\mathcal{O}(k^{2.5}k^{\sqrt{k}}n)$ time by applying the algorithm for Induced Max-Edge Biclique on bipartite graphs [78] which relies on the observation that one side has at most $\sqrt{k}$ vertices.

First, we study the parameterized complexity of enumerating all maximal non-induced bicliques in weakly γ-closed graphs. We need to define carefully, however, what we mean by enumerating bicliques: The algorithm of Eppstein [69] enumerates in $\mathcal{O}^*(2^d)$ time all maximal pairs of sets S and T such that each vertex of S is adjacent to each vertex of T. For this enumeration problem, an FPT-algorithm for the weak closure is unattainable since any clique of size n is 1-closed and admits $\Theta(2^n)$ bipartitions that need to be enumerated. To circumvent this issue, we view a biclique as a vertex set that can be partitioned into sets S and T. Thus, in order to strengthen the parameterization from d to γ, we go from an explicit listing of bicliques with bipartitions to a compact representation of bicliques as vertex sets and this is indeed necessary. We say that a vertex set $U \subseteq V(G)$ is a non-induced biclique if $G[U]$ contains a biclique as a (not necessarily induced) subgraph. Note that it can be decided in $\mathcal{O}(n^2)$ time whether a vertex set $U \subseteq V(G)$ is a non-induced biclique or not, because U is a non-induced biclique if and only if the complement of $G[U]$ has multiple connected components. We adapt the algorithm of Theorem 5.3 to obtain an $\mathcal{O}^*(2^\gamma)$-time algorithm to enumerate all maximal non-induced bicliques.

Recall that in Theorem 5.3 we bounded the overall number of maximal s-plexes in a weakly γ-closed graph G by distinguishing 4 different types of maximal s-plexes if we are provided wih the set of maximal s-plexes of $G - v$. As in the proof of Theorem 5.3, we aim to enumerate all maximal non-induced bicliques in G, provided with the collection $\mathcal{S}'$ of all non-induced maximal bicliques in $G' \coloneqq G - v$. Again, we define the same four types of non-induced bicliques S:

Type 1: S does not contain v.

Type 2: S contains v and $S \setminus \{v\}$ is maximal in G'.

Type 3: S contains v, $S \setminus \{v\}$ is not maximal in G', and S contains a non-neighbor of v.

Type 4: S contains v, $S \setminus \{v\}$ is not maximal in G', and S is contained in the neighborhood of v, that is, $S \subseteq N_G[v]$.

First and foremost, all maximal non-induced bicliques of Type 1 and Type 2 can be enumerated from $\mathcal{S}'$ in $|\mathcal{S}'| \cdot n^2$ time. We claim that there are at most $2^{\gamma-1}n$ maximal non-induced bicliques of Type 3: Let U be such a non-induced biclique with a bipartition (S, T). Without loss of generality, assume that $u, v \in S$. There are at most n choices for $u \in S \setminus N_G[v]$ and there are at most $2^{\gamma-1}$ choices for $T \subseteq N_G(v) \cap N_G(u)$. Since U is a maximal non-induced biclique, we obtain $S = \bigcap_{w \in T} N_G(w)$. Finally, there is only one maximal non-induced biclique of Type 4, namely $N_G[v]$. Thus, we obtain the following theorem.

Theorem 5.14. *All maximal non-induced bicliques can be enumerated in $\mathcal{O}^*(2^\gamma)$ time.*

Second, we consider the decision variant of this problem. We show that Non-Induced (k_1, k_2)-Biclique can be solved in $\mathcal{O}^*(2^\gamma)$ time, using this enumeration algorithm.

Theorem 5.15. Non-Induced (k_1, k_2)-Biclique *can be solved in $\mathcal{O}^*(2^\gamma)$ time.*

Proof. With the algorithm behind Theorem 5.14 we can enumerate the vertex sets of all maximal non-induced bicliques. This algorithm, however, only returns the vertex set, and not a bipartition of any maximal non-induced biclique. To check whether any of these maximal non-induced bicliques has a bipartition into sets S and T such that $|S| \geq k_1$ and $|T| \geq k_2$, we use the following observation: Let G' denote the complement graph of G. Any connected component of G' is either completely

contained in S or completely contained in T. Now, we can use this observation to define an instance of Subset Sum to check whether there exists a valid bipartition. Subset Sum is formally defined as follows.

Subset Sum
Input: A set $A = \{a_1, \ldots, a_n\}$ of n positive integers and $k_1 \leq k_2 \in \mathbb{N}$.
Question: Is there a set $B \subseteq A$ such that $k_1 \leq \sum_{b \in B} b \leq k_2$?

A standard dynamic programming algorithm can solve Subset Sum in $\mathcal{O}(n \cdot \sum_{a \in A} a)$ time. To solve Non-Induced (k_1, k_2)-Biclique, we construct an instance (A', k_1', k_2') of Subset Sum for each maximal non-induced biclique U with $|U| \geq k_1 + k_2$ returned by the algorithm of Theorem 5.14, where $k_1' := k_1$, $k_2' := |U| - k_2$, and $A' := \{|C_i| \colon i \in [\ell]\}$ for the connected components $C_1, \ldots, C_\ell \subseteq V(G)$ of the complement of $G[U]$. Observe that (G, k_1, k_2) is a Yes-instance if and only if the constructed instance of Subset Sum is a Yes-instance for some maximal non-induced biclique U: note that k_1' is a lower bound and k_2' is an upper bound for the size of the smaller side of any valid bipartition and any solution B of the Subset Sum instance corresponds to S, the smaller side of the bipartition of U, and $A \setminus B$ corresponds to the other part of the bipartition. □

Recall that Non-Induced Max-Edge Biclique can be solved by solving $\sqrt{k}$ instances of Non-Induced (k_1, k_2)-Biclique. Hence, we obtain the following from Theorem 5.15.

Corollary 5.16. Non-Induced Max-Edge Biclique *can be solved in* $\mathcal{O}^*(2^\gamma)$ *time.*

5.2.2 Induced Biclique

In this subsection, we study problems where one aims to find *induced* maximal bicliques fulfilling certain cardinality constraints. Gaspers et al. [90] provided an $\mathcal{O}^*(3^{n/3})$-time algorithm to enumerate all maximal induced bicliques. When $k_1 = k_2$ in Induced (k_1, k_2)-Biclique, we will refer to the problem as Induced (k, k)-Biclique. We also consider Induced Max-Edge Biclique where we demand that $|S| \cdot |T| \geq k$ instead of putting constraints on the partition sizes. Induced Max-Edge Biclique is NP-hard [194] and W[1]-hardness with respect to the solution size k can be shown by a reduction from Independent Set where we attach an universal vertex. As in the non-induced case, Induced Max-Edge Biclique can be solved by solving $\sqrt{k}$ instances of Induced (k_1, k_2)-Biclique. Thus, positive results for Induced (k_1, k_2)-Biclique transfer to Induced Max-Edge Biclique.

First, we present an FPT-algorithm for Induced (k,k)-Biclique parameterized by γ.

Theorem 5.17. Induced (k,k)-Biclique *can be solved in* $\mathcal{O}^*(\gamma^{\mathcal{O}(\gamma)})$ *time.*

Proof. Since a biclique $K_{\gamma,\gamma}$ is not weakly γ-closed, (G,k,k) is a No-instance if $k \geq \gamma$. Moreover, Induced (k,k)-Biclique is trivially solvable in polynomial time when $k \leq 1$. Hence, we may assume that $2 \leq k < \gamma$. Let σ be a fixed weak closure ordering of G. Suppose that (S,T) is a solution of (G,k). Furthermore, let $v \in S \cup T$ be the vertex of $S \cup T$ that appears in σ before all other vertices of $S \cup T$. We assume without loss of generality that v lies in S. Note that there are at most n choices for v. Let G' be the graph obtained by removing all vertices preceding v in σ. Furthermore, let $v' \in V(G') \setminus \{v\}$ be another vertex which is contained in S. Note that there are at most n choices for v'. Next, we determine an independent set $T \subseteq N_{G'}(v) \cap N_{G'}(v')$ of at least k vertices. Since $|N_{G'}(v) \cap N_{G'}(v')| < \gamma$, there are at most 2^γ possibilities for T. Now, it remains to find an independent set $S \subseteq \bigcap_{u \in T} N_{G'}(u)$ of size at least k in G'. In companion work, we showed that Independent Set admits a kernel with γk^2 vertices [137]. Hence, by brute-forcing on the resulting kernel, we can determine S in $\mathcal{O}^*((\gamma k^2)^k)$ time. Since $k < \gamma$, the overall running time is $\mathcal{O}^*(2^\gamma \gamma^{3\gamma}) = \mathcal{O}^*(\gamma^{\mathcal{O}(\gamma)})$. □

For c-closed graphs, we show that there is a single-exponential time algorithm when $k_1 \geq 2$. Our algorithm is based on a reduction to a variant of Independent Set called Bicolored Independent Set [52].

Bicolored Independent Set
Input: A graph G, a partition (V_1, V_2) of $V(G)$, and $k_1, k_2 \in \mathbb{N}$.
Question: Does there exist an independent set $I \subseteq V(G)$ with $|I \cap V_1| = k_1$ and $|I \cap V_2| = k_2$?

Theorem 5.18. *If* $k_1 \geq 2$*, then* Induced (k_1,k_2)-Biclique *can be solved in* $\mathcal{O}^*(1.611^c)$ *time.*

Proof. Let $(G = (V,E), k_1, k_2)$ be an instance of Induced (k_1,k_2)-Biclique. Since $k_1 \geq 2$ any induced biclique with k_1 vertices in one partite set and with k_2 vertices in the other partite set contains at least on cycle on four vertices. For each induced cycle (u_S, u_T, v_S, v_T) on four vertices in G we search the largest induced biclique containing these four vertices. Now, we construct an instance $(G', V_1', V_2', k_1, k_2)$ of Bicolored Independent Set, where

- $V_1' \coloneqq N_G(u_S) \cap N_G(v_S)$,

- $V_2' \coloneqq N_G(u_T) \cap N_G(v_T)$, and
- $G' \coloneqq (V_1' \cup V_2', E(G[V_1']) \cup E(G[V_2']) \cup \{v_1'v_2' \mid v_1' \in V_1', v_2' \in V_2', v_1'v_2' \notin E(G)\})$.

In other words, G' is constructed from $G[V_1' \cup V_2']$ by flipping the adjacency between V_1' and V_2'. By the c-closure of G, there are at most $2c-2$ vertices in G'. Since $v_1' \in V_1'$ and $v_2' \in V_2'$ are adjacent in G if and only if they are not in G', there is a (k_1, k_2)-biclique containing u_S, u_T, v_S, v_T if and only if $(G', V_1', V_2', k_1, k_2)$ is a Yes-instance. Since Bicolored Independent Set is $\mathcal{O}^*(1.2691^n)$-time solvable on n-vertex graphs [52], we obtain an $\mathcal{O}^*(1.611^c)$-time algorithm for Induced (k_1, k_2)-Biclique. □

By using a reduction similar to the one in the proof of Theorem 5.18, and using the algorithm of Gaspers et al. [90] to enumerate all maximal induced bicliques in $\mathcal{O}^*(3^{n/3})$ time we obtain the following.

Proposition 5.19. *All maximal induced bicliques in which each part has at least two vertices can be enumerated in $\mathcal{O}^*(3^{2c/3})$ time.*

However, even 2-closed graphs may have $\Omega(3^{n/3})$ maximal induced bicliques: Consider the aforementioned graph proposed by Hermelin and Manoussakis [111], which consists of a single universal vertex u and $(n-1)/3$ disjoint triangles. Observe that this graph is 2-closed and has $3^{(n-1)/3}$ maximal induced bicliques where one part consists of u.

In contrast to our positive result for $k_1 \geq 2$ presented in Theorem 5.18, we prove that Induced $(1, k)$-Biclique is NP-hard even on graphs with constant h-index, c-closure, and weak γ-closure.

Theorem 5.20. Induced Max-Edge Biclique *and* Induced $(1, k_2)$-Biclique *remain NP-hard even on graphs with h-index 4, c-closure 3, and weak γ-closure 2.*

Proof. We first show the NP-hardness for Induced Max-Edge Biclique. We reduce from Independent Set, which is NP-hard even on graphs in which each vertex has degree at most 3 [87]. Recall that in Independent Set we are given a graph G and an integer k, and ask whether G contains an independent set of size at least k. We assume that $k \geq 10$, since otherwise the instance (G, k) can be solved in polynomial time. We construct an instance (G', k') of Induced Max-Edge Biclique as follows: We begin with a copy of G. Then, each edge $uv \in E(G)$ is replaced by a path on four vertices u, u_v, v_u, and v. Finally, we introduce a new universal vertex w (that is, $N_{G'}[w] = V(G')$) and set $k' \coloneqq k + |E(G)|$. It is easy to see that G' has h-index 4 (because every vertex except w has degree at most 4), is 3-closed and weakly 2-closed. It remains to show that G contains an independent set

of size k if and only if G' contains an induced biclique with at least $k' = k + |E(G)|$ edges.

Suppose that G contains an independent set I of size at least k. Then, there is an independent set I' of size $k + |E(G)|$ in $G' - w$: Since I is an independent set, for each edge $uv \in E(G)$ we have without loss of generality that $u \notin I$. Let $F \coloneqq \{u_v \mid uv \in E(G) \text{ such that } u \notin I\}$ be the union of the neighbors of these vertices u not in the independent set in paths on four vertices in G'. Then, I' is the disjoint union of I and F. Thus, the set $I' \cup \{w\}$ is an induced biclique with at least $k + |E(G)|$ edges in G'.

Conversely, suppose that G' contains a biclique (S, T) with at least $k' = k+|E(G)|$ edges. Since each vertex in $G' - w$ has degree at most 3 and $k \geq 10$, we see that vertex w is contained in (S, T). Without loss of generality, assume that $w \in S$. Since w is a universal vertex, we obtain $S = \{w\}$. It follows that T is an independent set of size at least $k + |E(G)|$ in G'. We may assume $|T \cap \{u_v, v_u\}| = 1$: For each edge $uv \in E(G)$, the set T contains at most one of u_v and v_u. If neither is in T, then $(T \setminus \{u\}) \cup \{u_v\}$ is another independent set of size k'. Thus, we may assume that $|T \cap \{u_v, v_u\}| = 1$ for every $uv \in E(G)$. No pair of adjacent vertices u and v in G are part of T since otherwise T contains three vertices from a path (u, u_v, v_u, v). Thus, $T \cap V(G)$ is an independent set of size $|T'| - |E(G)| \geq k$.

Finally, note that this reduction also shows NP-hardness of Induced $(1, k_2)$-Biclique (let $k_2 = k'$). □

Together with Theorem 5.20, the next theorem paints a full picture of the complexity of Induced (k_1, k_2)-Biclique with respect to the weak closure number.

Theorem 5.21. *For constant $k_1 \geq 2$,* Induced (k_1, k_2)-Biclique *on weakly γ-closed graphs is polynomial-time solvable if $\gamma \leq k_1 + 1$ and NP-hard otherwise. Moreover,* Induced $(1, k_2)$-Biclique *on weakly 1-closed graphs is polynomial-time solvable.*

Proof. We start with the NP-hardness. We adapt the reduction in the proof of Theorem 5.20: Instead of adding a single universal vertex w, we add k_1 universal vertices (which are pairwise nonadjacent). Note that the graph constructed by our reduction is weakly $(k_1 + 2)$-closed (consider an ordering in which all the universal vertices appear last).

Our polynomial-time algorithms solve Independent Set on weakly 1-closed graphs as a subroutine. We fix a weak closure ordering σ. Start with $I = \emptyset$. In a first step, we add the last vertex v of σ to I and then delete $N[v]$ from the graph. For the correctness of this step, observe that the neighborhood of v is a clique. Otherwise,

there exists a non-neighbor u of v with $u <_\sigma v$ and distance 2 to v. Since u and v have at least one common neighbor, we obtain a contradiction to the fact that the graph is weakly 1-closed. Since $N[v]$ is a clique, there exists a maximum independent set containing v. We repeat this step until the graph is empty.

Next, we give a polynomial-time algorithm for Induced $(1, k_2)$-Biclique on weakly 1-closed graphs. Without loss of generality, we assume that the input graph G is connected. Observe that there is a universal vertex u that is adjacent to every other vertex. Now, observe that there is an induced $(1, k_2)$-biclique in G if and only if a maximum independent set of size k_2 in $G - u$. Since a maximum independent set in a weakly 1-closed graph can be found in polynomial time, we are done.

Finally, we prove the polynomial-time solvability for $\gamma \leq k_1 + 1$. Observe that if $\gamma \leq k_1$, then we have a No-instance of Induced (k_1, k_2)-Biclique since an induced (k_1, k_2)-biclique has weak closure $k_1 + 1$. Hence, in the following we assume that $\gamma = k_1 + 1$. Now, consider a hypothetical solution (S, T) with $|S| = k_1$ and $|T| = k_2$. We can guess which vertices correspond to the smaller side S in $\mathcal{O}(n^{k_1})$ time. Let σ be a fixed weak closure ordering and let X be the set of vertices that occur in σ before any vertex in S. Since $T \cap X$ are common neighbors of S we observe that $T \cap X$ has size at most $\gamma - 1 = k_1$. Hence, in $\mathcal{O}(n^{k_1})$ time, we can guess $T \cap X$. It remains to find $T \cap X$. Note that $T \cap X \subseteq U := \bigcap_{s \in S} N(s)$, that is, $S \subseteq N(t)$ for every vertex $t \in T \cap X$. Observe that $G[U \cap X]$ is weakly 1-closed: In the ordering σ, two nonadjacent vertices $u, u' \in U \cap X$ such that $u <_\sigma u'$ have no common neighbor $w \in U \cap X$ with $u <_\sigma w$ since u and u' have S as common neighbors which appear after u' in σ, and S has size $k_1 = \gamma - 1$. As argued above, we can find a maximum independent set in $G[U \cap X]$ in polynomial time. Thus, Induced (k_1, k_2)-Biclique can be solved in polynomial time if k_1 is a constant and $\gamma \leq k_1 + 1$. □

To complete the dichotomy with respect to c, we prove that Induced Max-Edge Biclique and Induced (k_1, k_2)-Biclique can be solved in polynomial time if $c = 2$. Observe that Theorem 5.18 implies a polynomial-time algorithm for $k_1 \geq 2$ if $c = 2$. Hence, it remains to show that Induced $(1, k_2)$-Biclique can be solved in polynomial-time if $c = 2$. For this, is it sufficient to consider diamond-free graphs since each 2-closed graph is diamond-free.

Proposition 5.22. Induced $(1, k_2)$-Biclique *can be solved in polynomial time on diamond-free graphs.*

Proof. Suppose that the input graph G is diamond-free. Then, for each vertex $v \in V(G)$ the graph $G[N(v)]$ is a disjoint union of cliques. Thus, $(G, 1, k_2)$ is a Yes-instance if and only if there is a vertex $v \in V(G)$ such that $G[N(v)]$ has at least k_2 connected components. □

Now, from Proposition 5.22 ($k_1 = 1$) and Theorem 5.18 ($k_1 \geq 2$) we obtain the following.

Corollary 5.23. Induced (k_1, k_2)-Biclique *and* Induced Max-Edge Biclique *can be solved in polynomial time on 2-closed graphs.*

Our results for Induced (k_1, k_2)-Biclique can be summarized as follows (see also Table 6.1): If $k_1 = k_2$, then the problem becomes FPT with respect to the weak closure number γ (Theorem 5.15). In the general case, the complexity strongly depends on whether $k_1 \geq 2$ or $k_1 = 1$. If $k_1 \geq 2$, the problem is polynomial-time solvable for $\gamma \leq k_1 + 1$ (Theorem 5.21), NP-hard for $\gamma \geq k_1 + 2$ (Theorem 5.21), and FPT for the parameterization by c (Theorem 5.18). If $k_1 = 1$, then we have a complexity dichotomies in terms of c and γ: we have a polynomial-time algorithm for $c = 2$ (Corollary 5.23) and $\gamma = 1$ (Theorem 5.21) and NP-hardness for $c \geq 3$ (Theorem 5.20) and $\gamma \geq 2$ (Theorem 5.20).

5.3 Conclusion

Fox et al. studied the complexity of enumerating maximal cliques in (weakly) closed graphs [84]. In this chapter, we continued this line of research and studied the complexity of enumerating maximal s-plexes (and other clique relaxations) or finding a sufficiently large s-plex (and other clique relaxations) in (weakly) closed graphs. More precisely, we studied in this chapter s-plexes, s-defective cliques, s-clubs, and (non-) induced bicliques.

For future work, it is interesting to study the parameterized complexity of further clique relaxations in (weakly) closed graphs in terms of enumeration or finding a solution of size at least k. One promising candidate are s-cliques. A vertex set S is an *s-clique* if each two vertices $u, v \in S$ have distance at most s in G [161]. Note that s-cliques are closely related to s-clubs; instead of requiring distance at most s in $G[S]$, here one requires distance at most s in G. It is known that the problem of finding an s-clique of size at least k admits an FPT-algorithm for $\Delta + s$ [143]. It is interesting whether this FPT result can be lifted to an FPT-algorithm with respect to $c + s$ or $d + s$.

Another interesting candidate are μ-complete vertex sets. A vertex set S is *μ-complete* for some $0 \leq \mu \leq 1$, if every vertex in $G[S]$ has degree at least $\mu \cdot (|S| - 1)$ [166, 195]. Especially the case $\mu \geq 1/2$ is interesting: The value of μ implies that $G[S]$ is connected, and even stronger, has diameter 2. In other words, S is also a 2-club. Since finding a sufficiently large 2-club is NP-hard even in 4-closed

graphs (Theorem 5.13), adding the additional constraint that each vertex has degree at least $\mu \cdot (|S| - 1)$ could help to obtain polynomial-time algorithms for finding sufficiently large μ-complete vertex sets in graphs with constant closure.

In Theorem 5.3, we showed that any weakly γ-closed graph has $\mathcal{O}(2^{\gamma} n^{2s-1})$ maximal s-plexes. Afterwards, we presented a graph with $\Omega((n/2)^{2s-2})$ maximal s-plexes. It is interesting to close this gap. For s-defective cliques the current gap between upper and lower bound is much smaller: In Theorem 5.6 we showed that any weakly γ-closed graph has $\mathcal{O}(2^{\gamma} n^{s+1})$ maximal s-defective cliques. Afterwards, we presented a graph with $\Omega((n/2)^{s+1})$ maximal s-defective cliques.

For s-Defective Clique we improved upon the dependence of s compared with the algorithm of enumerating all maximal s-defective cliques. More precisely, the enumeration algorithm implied that s-Defective Clique can be solved in $\mathcal{O}(2^{\gamma} n^{s+3})$ time. Furthermore, we present an algorithm which running-time is bounded by $2^{\mathcal{O}(\gamma\sqrt{s}+s\log k)} n^{\mathcal{O}(\sqrt{s})}$. The algorithm achieving this running-time relied on the fact that each s-defective clique can be covered by at most $\mathcal{O}(\sqrt{s})$ cliques. It is open, whether the exponent on n can be further reduced, for example to $1/4$. Another natural question is, whether such an improvement in terms of the dependence of s is also possible for s-Plex. Since an s-plex is not always coverable by $\mathcal{O}(\sqrt{s})$ cliques, our approach for s-defective cliques cannot be directly be adapted to s-plexes.

For 2-Club we provided NP-hardness even if $c = 4$ (Theorem 5.13). Afterwards, we argued that 4-closed graphs may have up to $3^{n/3}$ maximal 2-clubs. It remains an open question whether 2-Club can be solved in polynomial time if $c = 2$ or $c = 3$. Also, it remains open whether all maximal 2-clubs can be enumerated efficiently if $c = 2$ or $c = 3$. Furthermore, it is interesting to study the problem of finding a sufficiently large s-club or the enumerate all maximal s-clubs for $s \geq 3$ with respect to the $c$closure. It is very likely that our hardness results for 2-clubs can be lifted to larger values of s, for example by subdividing edges in the constructed instances.

Another interesting open question is whether our algorithm with running time $\mathcal{O}^*(\gamma^{\mathcal{O}(\gamma)})$ for Induced (k,k)-Biclique (Theorem 5.17) can be improved to an algorithm with running $\mathcal{O}^*(2^{\mathcal{O}(\gamma)})$. In the current algorithm, we determine one side of the sought biclique in $\mathcal{O}^*(2^{\gamma})$ time. For the other side, we rely on finding an independent set. This is done by applying brute-force on a kernel. To achieve the claimed running time, we have to improve the second step, where we determine this independent set. More precisely, if we would be able to find an independent set with $k \leq \gamma$ in $\mathcal{O}^*(2^{\mathcal{O}(\gamma)})$ time, then we would achieve the desired running time. The current fastest algorithm for this task is the aforementioned brute force algorithm on the kernel with up to γk^2 vertices [137].

In our algorithms we assume that the (weak) closure is given. To exploit this

value, the (weak) closure of the graph has to be computed first. The c-closure of a graph can be computed in $\mathcal{O}(n^\omega)$ time [84], where ω is the matrix multiplication coefficient or in $\mathcal{O}(cn^2 + m^{3/2})$ time [142]. An algorithm with the latter running time is preferable to the algorithm with running time $\mathcal{O}(n^\omega)$ if c is small and the graph is sparse. The computation of the weak closure γ is possible in $\mathcal{O}(n^3)$ time [84]. An important open question is thus whether the parameters c and γ can computed faster. Is the computation of these parameters doable in linear time?

Chapter 6

Complexity of s-Club with Triangle and Seed Constraints

A big drawback of s-clubs is that the largest s-clubs are often not very cohesive with respect to other cohesiveness measures such as density or minimum degree. Recall that a vertex set S of a graph G is an s-club if each pair of vertices in S has distance at most s in $G[S]$. For an example we refer to Figure 6.1. This behavior is particularly pronounced for $s = 2$: the largest 2-club in a graph is often the vertex v of maximum degree together with its neighbors [109]. Usually, this is not a good solution for applications since the number of edges within the induced subgraph is too small. To avoid these so-called hub-and-spoke structures, it has been proposed to augment the s-club definition by adding further constraints on the model [41, 145, 193, 222].

One of these augmented models, are the *vertex-triangle s-clubs*, proposed by Carvalho and Almeida [41].

Definition 6.1. A vertex set S in a graph G is a *vertex-triangle s-club* if S is an s-club and each vertex in S is part of at least one triangle in $G[S]$.

The property is motivated by the importance of triangles, or clusters of size 3, since for example, the definitions of the global and local clustering coefficient are based on triangles [2, 179, 225]. Since each vertex in a vertex-1-triangle s-club S is contained in at least one triangle, S contains at least $\lceil |S|/3 \rceil$ triangles. In other words, the triangle property ensures that there are many edges within the induced subgraph $G[S]$. Vertex-triangle s-clubs relax cliques also in the following sense: In a clique C of size c each vertex in C is contained in exactly $\binom{c-1}{2}$ triangles and in a vertex-triangle s-club each vertex is contained in at least one triangle.

The vertex-triangle s-club property was later generalized to the *vertex-ℓ-triangle* property [3].

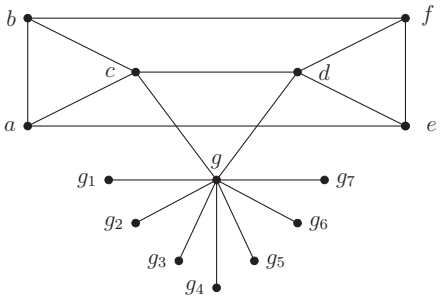

Figure 6.1: In this graph the largest 2-club is $\{c, d, g, g_1, \ldots, g_7\}$, the largest vertex-1-triangle 2-club is $\{a, b, c, d, e, f, g\}$, and the largest edge-1-triangle 2-club is $\{c, d, e, f, g\}$.

Definition 6.2. A vertex set S in a graph G is a *vertex-ℓ-triangle s-club* if S is an s-club and each vertex in S is part of at least ℓ triangle in $G[S]$.

This model is motivated by cliques relaxation based on clustering coefficients, so-called local- and global-α clusters, proposed by Ertem et al. [72]. Furthermore, such a property leads to higher clustering coefficients. For an example we refer to Figure 6.1.

Vertex Triangle s-Club
Input: An undirected graph $G = (V, E)$, and two integers $k, \ell \geq 1$.
Question: Does G contain an s-club S of size at least k that fulfills the vertex-ℓ-triangle property?

The vertex-ℓ-triangle constraint entails some desirable properties for cohesive subgraphs. Let D_v be the set of vertices which are in a triangle with $v \in V(G)$. Since $|D_v| \geq \sqrt{2\ell}$ we obtain that the minimum degree of a vertex-ℓ-triangle s-club is larger than $\sqrt{2\ell}$. Moreover, vertex-ℓ-triangle s-clubs S are robust with respect to some specific edge-deletions: Since $|D_v| \geq \sqrt{2\ell}$ we may delete up to $\sqrt{2\ell} - 1$ edges with one endpoint being v and the other in D_v without disconnecting S. This property is *not* fulfilled, if we instead of the vertex-ℓ-triangle constraint put the additional property of having minimum degree at least $\sqrt{2\ell}$ on s-clubs: In such a model there exists at least one vertex v such that each each deletion incident with v disconnects the graph, for example, a star with $\sqrt{2\ell}$ leaves, where a clique

Figure 6.2: The sets C_1, C_2 are cliques of size d. Hence, $C_1 \cup C_2$ is a vertex-$\binom{d}{2}$-triangle 3-club of size $2d$. After deleting the edge uv the cliques C_1 and C_2 are disconnected.

of size $\sqrt{2\ell}$ is attached to each leaf. However, some undesirable behavior of hub-and-spoke structures remains: vertex-ℓ-triangle s-clubs are not robust with respect to each possible edge deletion. For an example, we refer to Figure 6.2.

To overcome this problem, we introduce a new model where we put triangle constraints on the edges of the s-club instead of the vertices.

Definition 6.3. A vertex set S of a graph G fulfills the *edge-ℓ-triangle* property if $G[S]$ contains a spanning subgraph $G' \coloneqq (S, E')$ such that every edge in $E(G')$ is in at least ℓ triangles in G' and the diameter of G' is at most s.

For an example we refer to Figure 6.1. Next, we introduce the related problem.

Edge Triangle s-Club
Input: An undirected graph $G = (V, E)$, and two integers $k, \ell \geq 1$.
Question: Does G contain a vertex set S of size at least k that fulfills the edge-ℓ-triangle property?

Note that in this definition, the triangle and diameter constraints are imposed on a spanning subgraph of $G[S]$. In contrast, for Vertex Triangle s-Club, they are imposed directly on $G[S]$. The reason for this distinction is that we would like to have properties that are closed under edge insertions. Properties which are closed under edge insertions are also well-motivated from an application point of view since adding a new connection within a group should not destroy this group. If we would impose the triangle constraint on the induced subgraph $G[S]$ instead, then an edge-ℓ-triangle s-club S would not be robust to edge additions. For an example, we refer to Figure 6.3. By searching for a subgraph we can overcome this issue since this allows us to ignore such edges.

Observe that every set that fulfills the edge-ℓ-triangle property also fulfills the vertex-ℓ-triangle property. Also note that the converse is not true: A vertex-ℓ-triangle s-club is not necessarily also an edge-ℓ-triangle s-club. For example, the graph shown in Figure 6.2 fulfills the vertex-1-triangle property but does *not* fulfill the edge-1-triangle property since uv is contained in no triangle. Moreover, each

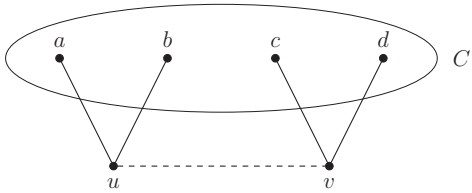

Figure 6.3: The set C induces a clique of size at least four. $S \coloneqq V(G) = C \cup \{u, v\}$ is a 3-club and every edge in $G[S]$ is contained in at least one triangle. Let G' be the graph obtained from G by adding the dashed edge uv. Since uv is not contained in any triangle in G', the set S would not be an edge-1-triangle 3-club in G' if we would impose the triangle constraints on $G'[S]$. In contrast, since we impose the triangle constraints on a spanning subgraph of $G'[S]$, the set S is an edge-1-triangle 3-club in G'.

vertex $v \in S$ has at least $\ell + 1$ neighbors in S: Consider an arbitrary edge uv. Since uv is in at least ℓ triangles $\{u, v, w_1\}, \ldots, \{u, v, w_\ell\}$ we thus conclude that u and v have degree at least ℓ. We can show an even stronger statement: an edge-ℓ-triangle s-club S is robust against up to ℓ edge deletions, as desired.

Proposition 6.4. *Let $G = (V, E)$ be a graph and let S be an edge-ℓ-triangle s-club in G. Let G' be a spanning subgraph of $G[S]$ such that every edge in G' is in at least ℓ triangles and the diameter of G' is at most s. If ℓ edges of G' are removed from G, then S is still an $(s + \ell)$-club and a $(2s)$-club in G.*

Proof. We show that if ℓ edges are removed from G', the diameter of the resulting graph $\widetilde{G}$ increases by at most ℓ. Let $P = (v_1, \ldots, v_{s+1})$ be a path of length s in G'. Since G' is an edge-ℓ-triangle s-club, every edge $v_i v_{i+1}$ of P is part of at least ℓ triangles in G'. Thus, for two vertices v_i and v_{i+1} in P there is a path of length at most two from v_i to v_{i+1} in G', either directly through the edge $v_i v_{i+1}$ or via a vertex u that forms one of the ℓ triangles with v_i and v_{i+1} in G'. Thus, $\operatorname{dist}(v_i, v_{i+1})$ increases by at most 1 after one edge deletion and only if $v_i v_{i+1}$ is removed. Since at most ℓ of the edges in P are removed, we have $\operatorname{dist}(v_1, v_{s+1}) \leq \operatorname{dist}(v_1, v_2) + \ldots + \operatorname{dist}(v_s, v_{s+1}) \leq s + \ell$ in $\widetilde{G}$. By the same arguments, we also have $\operatorname{dist}(v_1, v_{s+1}) \leq 2s$.

Thus, after deleting ℓ edges in G', S is an $(s + \ell)$-club and a $(2s)$-club. □

We also introduce and study a further variant of s-clubs. This variant of s-Club is also practically motivated but not necessarily by concerns about the robustness of the s-club. Here the difference to the standard problem is simply that we are given a set of seed vertices W and aim to find a large s-club that contains all seed vertices.

Seeded s-Club
Input: An undirected graph $G = (V, E)$, a subset $W \subseteq V$, and an integer $k \geq 1$.
Question: Does G contain an s-club S of size at least k such that $W \subseteq S$?

This variant can be used in community detection, where we are often interested in finding communities containing some set of fixed vertices [124, 230].

In this chapter, we study the parameterized complexity of the three above-mentioned problems with respect to the standard parameter solution size k. Our goal is to determine whether FPT results for s-Club [43, 206] transfer to these practically motivated problem variants.

Related Work. The s-Club problem is NP-hard for all $s \geq 1$ [31], even when the input graph has diameter $s+1$ [14]. For $s = 1$, s-Club is equivalent to Clique and thus W[1]-hard with respect to k. In contrast, for every $s > 1$, s-Club is fixed-parameter tractable (FPT) with respect to the solution size k [43, 206]. This fixed-parameter tractability can be shown via a Turing kernel with $\mathcal{O}(k^2)$ vertices for even s and $\mathcal{O}(k^3)$ vertices for odd s [43, 206]. These Turing kernels are based on the observation that each vertex v together which each other vertex w with distance at most $\lfloor s/2 \rfloor$ is an s-club.

The complexity of s-Club has been also studied with respect to different classes of input graphs [93] and with respect to structural parameters such as degeneracy of the input graph [110]. The s-Club problem can be solved efficiently in practice, in particular for $s = 2$ [31, 43, 109].

Vertex Triangle s-Club is NP-hard for all interesting cases, that is, $s \geq 2$ and for all $\ell \geq 1$ [41, 3]. Furthermore, there exist ILP formulations for Vertex Triangle s-Club [41, 3]. We are not aware of any algorithmic studies of Edge Triangle s-Club or Seeded s-Club. The NP-hardness of Edge Triangle s-Club for $\ell = 1$ can be shown via the reduction for Vertex Triangle s-Club for $\ell = 1$ [3]. Also, the NP-hardness of Seeded s-Club follows directly from the fact that an algorithm for the case where $|W| = 1$ can be used as a black box to solve s-Club.

Further robust models of s-clubs include t-hereditary s-clubs [145, 193], t-robust s-clubs [222], and t-connected s-clubs [231, 145]. Note that these 3 models do not guarantee triangles and thus the cluster coefficients of the solutions of these models might be 0.

Our Results. An overview of our results is given in Table 6.1. For Vertex Triangle s-Club and Edge Triangle s-Club, we provide a complexity dichotomy

for all interesting combinations of s and ℓ, that is, for every $s \geq 2$ and $\ell \geq 1$, into cases that are FPT or W[1]-hard with respect to k, respectively. Our W[1]-hardness reduction for Edge Triangle s-Club for $\ell \geq 2$ also shows the NP-hardness of this case. The FPT-algorithms are obtained via adaptions of the Turing kernelization for s-Club. Interestingly, Vertex Triangle s-Club with $\ell = 1$ is FPT only for larger s, whereas Edge Triangle s-Club with $\ell = 1$ is FPT for all s. In our opinion, this means that the edge-ℓ-triangle property is preferable not only from a modelling standpoint but also from an algorithmic standpoint as it allows to employ Turing kernelization as a part of the solving procedure, at least for $\ell = 1$. It is easy to see that standard problem kernels of polynomial size are unlikely to exist for Vertex Triangle s-Club and Edge Triangle s-Club: s-clubs are necessarily connected and thus taking the disjoint union of graphs gives a trivial or-composition (see Section 2.3 for the definition) and, consequently a polynomial problem kernel implies coNP $\subseteq$ NP/poly [24].

All of our hardness results for Vertex Triangle s-Club and Edge Triangle s-Club are shown by a reduction from Clique which is W[1]-hard with respect to k [54, 63]. The idea is to replace each vertex of the Clique instance by a vertex gadget. These gadgets are constructed in such a way that if one of these vertices is part of a vertex/edge-ℓ-triangle 2-club S, then the entire vertex gadget is part of S. We can then use the distance constraint to make sure that full vertex gadgets are chosen only if the corresponding vertices are adjacent. While this idea is very natural, using the triangle constraint without creating many vertices that are too close to each other turned out to be technically challenging.

For Seeded s-Club, we provide a kernel with respect to k for clique seeds W and W[1]-hardness with respect to k for some other cases. For $s = 2$, our results provide a dichotomy into FPT and W[1]-hardness with respect to k in terms of the structure of the seed.

The W[1]-hardness of Seeded s-Club is provided by two reductions from the Clique problem. One reduction is for the case $s = 2$ and any seed that contains at least two non-adjacent vertices u and z. The other reduction is for the case $s \geq 3$ and any seed that contains at least two connected components U and Z. In both cases we add two copies X_1 and X_2 of the graph of the Clique instance to the new instance of Seeded s-Club such that each vertex of X_1 has distance at most s to u or U if its copy in X_2 is also part of the solution. We show a similar property for X_2 and z or Z. This ensures that if $p_1 \in X_1$ is in the solution if and only if $p_2 \in X_2$ is in the solution. Also, the reductions have the property that vertex p_1 in X_1 has distance at most s to vertex q_2 in X_2 if and only if pq is an edge of the Clique instance. This feature ensures that the same clique has to be chosen from both copies.

Table 6.1: Overview of our results of the parameterized complexity of the three problems with respect to the parameter solution size k.

	Vertex Triangle s-Club	Edge Triangle s-Club	Seeded s-Club
FPT	$\ell = 1$ and $s \geq 4$	$\ell = 1$ for each s	$G[W]$ is a clique
W[1]-h	$\ell = 1$ and $s \leq 3$	$\ell \geq 2$ for each s	$s = 2$ and $G[W]$ contains at least two non-adjacent vertices
	$\ell \geq 2$ for each s		$s \geq 3$ and $G[W]$ contains at least two connected components

Our W[1]-hardness results, in particular those for Seeded s-Club, show that the FPT results for s-Club are quite brittle since the standard argument that we may assume $k \geq \Delta$ fails and that adding even simple further constraints makes finding small-diameter subgraphs much harder.

6.1 Vertex Triangle s-Club

In this section, we settle the parameterized complexity of Vertex Triangle s-Club with respect to the solution size k. First, we show that this problem is fixed-parameter tractable when $\ell = 1$ and $s \geq 4$. Afterwards, we show W[1]-hardness for all remaining cases, that is, for $\ell \geq 2$ and $s \geq 2$, and also for $\ell = 1$ and $s \in \{2, 3\}$.

6.1.1 FPT-Algorithms

The overall idea is based on the idea of the Turing kernel for 2-Club, that is, bounding the size of $N_s[v]$ for each vertex $v \in V(G)$. The first step is to remove all vertices which are not in a triangle.

Reduction Rule 6.1. *Let (G, k) be an instance of* Vertex Triangle s-Club. *Delete all vertices from G which are not part of any triangle.*

Clearly, Reduction Rule 6.1 is correct and can be exhaustively applied in polynomial time. The application of Reduction Rule 6.1 has the following effect: if some vertex v is close to many vertices, then (G, k) is a trivial yes-instance.

Lemma 6.5. *Let (G, k) be an instance of* Vertex Triangle s-Club *with $\ell = 1$ and $s \geq 4$ to which Reduction Rule 6.1 is applied. Then, (G, k) is a yes-instance if $|N_{\lfloor s/2 \rfloor - 1}[v]| \geq k$ for some vertex $v \in V(G)$.*

Proof. Let $v \in V(G)$ be a vertex such that $|N_{\lfloor s/2 \rfloor - 1}[v]| \geq k$. We construct a vertex-1-triangle s-club T of size at least $|N_{\lfloor s/2 \rfloor - 1}[v]| \geq k$. Initially, we set $T \coloneqq N_{\lfloor s/2 \rfloor - 1}[v]$. Now, for each vertex $w \in N_{\lfloor s/2 \rfloor - 1}(v)$ we do the following: Since Reduction Rule 6.1 is applied, we conclude that there exist two vertices x and y such that $G[\{w, x, y\}]$ is a triangle. We add x and y to the set T. We call the set of vertices added in this step the *T-expansion*.

Next, we show that T is indeed a vertex-1-triangle s-club for $s \geq 4$. Observe that each vertex in T is either in $N_{\lfloor s/2 \rfloor - 1}[v]$ or a neighbor of a vertex in $N_{\lfloor s/2 \rfloor - 1}(v)$. Hence, each vertex in T has distance at most $\lfloor s/2 \rfloor$ to vertex v. Thus, T is an s-club. It remains to show that each vertex of T is in a triangle. Observe that for each vertex $w \in N_{\lfloor s/2 \rfloor - 2}[v]$ we have $N(w) \subseteq N_{\lfloor s/2 \rfloor - 1}[v]$. Recall that since Reduction Rule 6.1 is applied, each vertex in G is contained in a triangle. Thus, each vertex of $N_{\lfloor s/2 \rfloor - 2}[v]$ is contained in a triangle in T. Furthermore, all vertices in $N_{\lfloor s/2 \rfloor - 1}(v) \cup (T \setminus N_{\lfloor s/2 \rfloor - 1}[v])$ are in a triangle because of the T-expansion. Since $|T| \geq |N_{\lfloor s/2 \rfloor - 1}[v]| \geq k$, the statement follows. □

Next, we show that Lemma 6.5 implies the existence of a Turing kernel for $s \geq 4$. We do this by showing that $N_s[v]$ is bounded for every vertex in the graph. This in turn implies that the problem is fixed-parameter tractable. It is sufficient to bound the size of $N_s[v]$ for each $v \in V(G)$ since we then can query the oracle for an s-club of size k.

Theorem 6.6. Vertex Triangle s-Club *for $\ell = 1$ admits a k^4-vertex Turing kernel if $s = 4$ or $s = 7$, a k^5-vertex Turing kernel if $s = 5$, and a k^3-vertex Turing kernel if $s = 6$ or $s \geq 8$.*

Proof. First, we apply Reduction Rule 6.1. Because of Lemma 6.5 we conclude that (G, k) is a trivial yes-instance if $|N_{\lfloor s/2 \rfloor - 1}[v]| \geq k$ for any vertex $v \in V(G)$. Thus, in the following we can assume that $|N_{\lfloor s/2 \rfloor - 1}[v]| < k$ for each vertex $v \in V(G)$. We use this fact to bound the size of $N_s[v]$ in non-trivial instances.

Note that if $s = 4$ or $s = 5$ we have $\lfloor s/2 \rfloor - 1 = 1$. In this cases from $|N_{\lfloor s/2 \rfloor - 1}[v]| < k$ we obtain that the size of the neighborhood of each vertex is bounded. Thus, we obtain a k^4-vertex Turing kernel for $s = 4$ and a k^5-vertex Turing kernel for $s = 5$. Furthermore, if $s = 7$ we have $\lfloor s/2 \rfloor - 1 = 2$. Thus, we obtain a k^4-vertex Turing kernel for $s = 7$ since $N_7[v] \subseteq N_8[v] = N_2[N_2[N_2[N_2[v]]]]$.

If $s = 6$ or $s \geq 8$, then $\lfloor s/2 \rfloor - 1 \geq \lceil s/3 \rceil$. Observe that $N_s[v]$ is contained in $N_{\lceil s/3 \rceil}[N_{\lceil s/3 \rceil}[N_{\lceil s/3 \rceil}[v]]]$. Thus, we obtain a k^3-vertex Turing kernel for $s = 6$ or $s \geq 8$. □

Note that $s \geq 4$ is necessary to ensure $\lfloor s/2 \rfloor - 1 \geq 1$. In our arguments to obtain a Turing kernel $\ell = 1$ is necessary for the following reason: if $\ell \geq 2$, then the remaining vertices of the other triangles of a vertex in the T-expansion may be contained in $N_{\lfloor s/2 \rfloor + 1}$ and, thus, adding them will not necessarily give an s-club. Also note that using $N_t[v]$ for some $t < \lfloor s/2 \rfloor - 1$ does not help: The remaining vertices of the other triangles of a vertex in the T-expansion may be contained in N_{t+1}. But now, another T-expansion for the vertices in N_{t+1} is necessary. This may lead to a cascade of T-expansions where eventually we add vertices with distance at least $s+1$ to v. Thus, the constructed set is no s-club anymore.

6.1.2 Parameterized Hardness

In the following, we prove W[1]-hardness for Vertex Triangle s-Club parameterized by the solution size k for all cases not covered by Theorem 6.6, that is, $\ell \geq 2$ and $s \geq 2$, and also for $\ell = 1$ and $s \in \{2, 3\}$.

Theorem 6.7. Vertex Triangle s-Club *is W[1]-hard for parameter k if $\ell \geq 2$, and if $\ell = 1$ and $s \in \{2, 3\}$.*

For some combinations of s and ℓ we provide hardness for restricted input graphs. More precisely, we prove that Vertex Triangle s-Club is W[1]-hard even if each vertex $v \in V(G)$ is contained in *exactly* ℓ triangles in the input graph. In other words, the hardness does not depend on the fact that we could choose different triangles. We provide this hardness for the case $s \geq 3$ and arbitrary ℓ and also for $s = 2$ when $\ell = \binom{c-1}{2}$ for some integer c.

We prove the theorem by considering four subcases. The proofs for the four cases all use a reduction from the W[1]-hard Clique problem. In these constructions, each vertex v of the Clique instance is replaced by a vertex gadget T^v such that every vertex-ℓ-triangle s-club S either contains T^v completely or contains no vertex of T^v. This property is obtained since each vertex in T^v will be in exactly ℓ triangles and each of these triangles is within T^v. The idea is that if $uv \notin E(G)$ then there exists a vertex $x \in T^u$ and a vertex $y \in T^v$ such that $\text{dist}(x, y) \geq s + 1$.

Vertex Triangle 2-Club. First, we handle the case $s = 2$.

Construction 6.8. Let (G, k) be an instance of Clique and let c be the smallest number such that $\binom{c-1}{2} \geq \ell$. We construct an instance $(G', c(k+1), \ell)$ of Vertex Triangle 2-Club as follows.

- For each vertex $v \in V(G)$, we add a clique $T^v \coloneqq \{x_1^v, \ldots, x_c^v\}$ of size c to G'.
- For each edge $vw \in E(G)$, we connect the cliques T^v and T^w by adding the edge $x_{2i-1}^v x_{2i}^w$ and $x_{2i-1}^w x_{2i}^v$ to G' for each $i \in [\lfloor c/2 \rfloor]$.
- Furthermore, we add a clique $Y \coloneqq \{y_1, \ldots, y_c\}$ of size c to G'.
- We also add, for each $i \in [c]$ and each $v \in V(G)$, the edge $x_i^v y_i$ to G'.

Note that the clique size c ensures that each vertex $x \in V(G')$ is contained in at least $\binom{c-1}{2} \geq \ell$ triangles in G'. Furthermore, note that the clique Y is only necessary when c is odd to ensure that the vertices x_c^v and x_c^w have a common neighbor. We add the clique Y in both cases to unify the construction and the correctness proof. Next, we show that for each vertex gadget T^v the intersection with each vertex-ℓ-triangle 2-club is either empty or T^v.

Lemma 6.9. *Let S be a vertex-ℓ-triangle 2-club in G'. Then,*

a) $S \cap T^v \neq \emptyset \Leftrightarrow T^v \subseteq S$, and

b) $S' \coloneqq S \cup Y$ is also a vertex-ℓ-triangle 2-club in G'.

Proof. First, we show statement *a*). Assume that for a vertex $z \in T^v$ for some $v \in V(G)$ we have $z \in S$ for some vertex-ℓ-triangle 2-club S. Note that T^v contains all vertices which form a triangle with vertex z. Since c is minimal such that $\binom{c-1}{2} \geq \ell$ and since T^v is a clique, we conclude that $T^v \subseteq S$ to fulfill the property that vertex z is contained in at least ℓ triangles in $G[S]$. Thus, $T^v \subseteq S$.

Second, we show statement *b*). Since each vertex $y \in Y$ forms only triangles with vertices in Y and Y has size $\binom{c-1}{2}$, we conclude that $Y \subseteq S^* \Leftrightarrow S^* \cap Y \neq \emptyset$ for each vertex-ℓ-triangle 2-club S^*. In the following, let S be a vertex-ℓ-triangle 2-club such that $Y \cap S = \emptyset$. From statement *a*) we conclude that $S \coloneqq \bigcup_{v \in P} T^v$ for some set $P \subseteq V(G)$. Next, we show that $S' \coloneqq S \cup Y$ is also a vertex-ℓ-triangle 2-club. Since each vertex is contained in a clique of size c in S, it is in sufficiently many triangles. Thus, it remains to prove that S' is a 2-club. Consider some vertex x_i^v and some vertex y_j for some $i, j \in [c]$ and $v \in P$. Then, y_i is a common neighbor of x_i^v and y_j. Hence, S' is a vertex-ℓ-triangle 2-club and thus *b*) holds. □

Now, we prove the correctness of Construction 6.8.

Lemma 6.10. *For each $\ell \in \mathbb{N}$, the* Vertex Triangle 2-Club *problem parameterized by k is W[1]-hard.*

Proof. We prove that G contains a clique of size k if and only if G' contains a vertex-ℓ-triangle 2-club of size at least $c(k+1)$.

Let C be a clique of size at least k in G. We argue that $S \coloneqq Y \cup \bigcup_{v \in C} T^v$ is a vertex-ℓ-triangle 2-club of size $c(k+1)$ in G'. Note that for each vertex $v \in T^v$ we have $|T^v| = c$. Since T^v is a clique, we conclude that each vertex in T^v is contained in exactly $\binom{c-1}{2} \geq \ell$ triangles. The same is true for each vertex in Y. Hence, each vertex in S is contained in at least ℓ triangles. Thus, it remains to show that S is a 2-club. Consider the vertices x_i^v and x_j^w for $v, w \in C$, $i \in [c-1]$, and $j \in [c]$. If i is odd, then $x_{i+1}^w \in N(x_i^v) \cap N(x_j^w)$. Otherwise, if i is even, $x_{i-1}^w \in N(x_i^v) \cap N(x_j^w)$. In both cases, we obtain $\operatorname{dist}(x_i^v, x_j^w) \leq 2$. Next, consider two vertices x_c^v and x_c^w in S. Observe that $y_c \in N(x_c^v) \cap N(x_c^w)$. Since Y is a clique, it remains to consider vertices x_i^v and y_j in S for $i \in [c]$ and $j \in [c]$. Observe that $x_j^v \in N(y_j) \cap N[x_i^v]$. Thus, G' contains a vertex-ℓ-triangle 2-club of size at least $c(k+1)$.

Conversely, suppose that G' contains a vertex-ℓ-triangle 2-club S of size at least $c(k+1)$. By Lemma 6.9, we can assume that $Y \subseteq S$ and for each vertex gadget $T^v \in G'$ we either have $T^v \subseteq S$ or $T^v \cap S = \emptyset$. Hence, S contains at least k cliques of the form T^v. Assume towards a contradiction that S contains two cliques T^v and T^w such that $vw \notin E(G)$ and consider the two vertices $x_1^v \in T^v$ and $x_2^w \in T^w$. Note that these vertices always exist since $c \geq 3$. Observe that $N[x_1^v] = T^v \cup \{x_2^u \mid uv \in E(G)\} \cup \{y_1\}$ and $N[x_2^w] = T^w \cup \{x_1^u \mid uw \in E(G)\} \cup \{y_2\}$. Thus, $\operatorname{dist}(x_1^v, x_2^w) \geq 3$, a contradiction. Hence, for each two distinct vertex gadgets T^v and T^w that are contained in S, we observe that $vw \in E(G)$. Consequently, the set $\{v \mid T^v \subseteq S\}$ is a clique of size at least k in G. □

If $\ell = \binom{c-1}{2}$ for some integer c, then Lemma 6.10 also holds for the restriction that each vertex is contained in exactly ℓ triangles in the input graph G'.

Vertex Triangle s-Club for $s = 3$ and for $s \geq 4$ and $\ell \geq 2$. Now, we provide hardness for the remaining cases. We consider three subcases. Case 1 deals with odd s. Case 2 covers the case that s is even and $\ell \geq 3$. Case 3 deals with the case that s is even and $\ell = 2$. All three cases use the same vertex gadget. Only the edges between these gadgets, called *connector edges*, differ. The idea is to construct the vertex gadgets T^v in such a way that there are pairs of vertices in T^v of distance $2s^*$ which is almost s. Thus, in a vertex-ℓ-triangle s-club, the distance between two different vertex gadgets must be small. The first part of the following construction

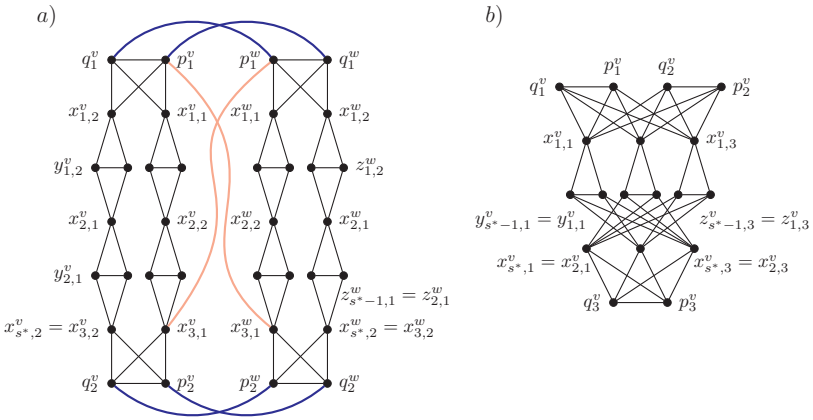

Figure 6.4: *a*) The vertex gadgets T^v and T^w for $s \in \{7, 8\}$ and $\ell = 2$. The blue lines are only added if s is odd and the red lines are only added if s is even. *b*) The vertex gadget T^v for $s \in \{5, 6\}$ and $\ell = 3$.

describes the vertex gadget which is used in all three cases. For an illustration of this construction see Figure 6.4.

Construction 6.11. We set $s^* \coloneqq \lfloor (s-1)/2 \rfloor$. Let (G, k) be an instance of Clique. We construct an equivalent instance $(G', 3\ell k s^*, \ell)$ of Vertex Triangle s-Club. Recall that $\ell \geq 1$ and $s = 3$, or $\ell \geq 2$ and $s \geq 4$. For each vertex $v \in V(G)$ we construct a vertex gadget T^v. This vertex gadget is used for each reduction of the three cases.

- We add the vertices p_i^v, and q_i^v, and an edge $p_i^v q_i^v$ for each $i \in [\ell]$ to G'.
- We add a vertex $x_{j,i}^v$ for each $i \in [\ell]$ and each $j \in [s^*]$ to G'.
- We add an edge $y_{t,i}^v z_{t,i}^v$ for each $i \in [\ell]$ and each $t \in [s^* - 1]$ to G'. Note that these vertices only exist, if $s \geq 5$.

The vertices $x_{j,i}^v$, $y_{t,i}^v$, and $z_{t,i}^v$ are referred to as the *cascading vertices*. They are used to ensure that all vertices in T^v are in exactly ℓ triangles and that there are vertex pairs of distance $2s^*$ within T^v. Note that since $s \geq 3$ we create at least ℓ many x-vertices. We connect these vertices as follows:

- We add the edges $p_i^v x_{1,j}^v$ and $q_i^v x_{1,j}^v$ for each $i \in [\ell - 1]$ and each $j \in [\ell]$ to G'.
- We add the edges $p_\ell^v x_{s^*,j}^v$ and $q_\ell^v x_{s^*,j}^v$ for each $j \in [\ell]$ to G'.
- We add the edges $y_{t,i}^v x_{t,i}^v$ and $z_{t,i}^v x_{t,i}^v$ for each $i \in [\ell]$, and each $t \in [s^* - 1]$ to G'.
- We add the edges $y_{t,i}^v x_{t+1,j}^v$ and $z_{t,i}^v x_{t+1,j}^v$ for each $i \in [\ell]$, each $j \in [\ell] \setminus \{i\}$, and each $t \in [s^* - 1]$ to G'.

Note that if $s = 3$ or $s = 4$, the graph G' is a non-induced biclique where one partite set consists of the vertices $\{x_{1,j}^v \mid j \in [\ell]\}$ and the other partite set consists of the vertices $\{p_i^v, q_i^v \mid i \in [\ell]\}$. Furthermore, the additional edges are $p_i^v q_i^v$ for each $i \in [\ell]$. Also, observe that each vertex gadget T^v consists of exactly $3\ell s^*$ vertices.

From now on, the construction differs for the three cases. We now connect these vertex gadgets by introducing the *connector edges*: For each edge $vw \in E(G)$ we add edges between the vertex gadgets T^v and T^w of the corresponding vertices. We distinguish between the three cases.

Case I: s **is odd.** We add the edges $p_i^v q_i^w$ and $q_i^v p_i^w$ for each $i \in [\ell]$ to G'.

Case II: s **is even and** $\ell \geq 3$**.** We add the edges $p_1^v q_1^w$, $q_1^v p_1^w$, $p_\ell^v q_\ell^w$, and $q_\ell^v p_\ell^w$ to G'.

Case III: s **is even and** $\ell = 2$**.** We add the edges $p_1^v x_{s^*,1}^w$, and $p_1^w x_{s^*,1}^v$ to G'.

We make the following observation about the connector edges between different vertex gadgets: If s is odd (Case I), or if s is even and $\ell \geq 3$ (Case II), we have $N(p_i^v) \setminus T^v \subseteq \{q_i^w \mid vw \in E(G)\}$ and also $N(q_i^v) \setminus T^v \subseteq \{p_i^w \mid vw \in E(G)\}$ for each $v \in V(G)$ and each $i \in [\ell]$. Otherwise, if s is even and $\ell = 2$ (Case III), observe that we have $N(p_1^v) \setminus T^v = \{x_{s^*,1}^w \mid vw \in E(G)\}$ and also $N(x_{s^*,1}^v) \setminus T^v = \{p_1^w \mid v^w \in E(G)\}$ for each $v \in V(G)$. Since only the connector edges have endpoints in different vertex gadgets, we thus observe the following.

Observation 6.12. *All three endpoints of each triangle in G' are contained in exactly one vertex gadget.*

Next, we show that each vertex in a vertex gadget T^v is contained in exactly ℓ triangles in $G'[T^v]$. Together with Observation 6.12 this implies that each vertex in G' is contained in exactly ℓ triangles and also that for each vertex gadget T^v any vertex-ℓ-triangle s-club S in G' either contains all vertices of T^v or none of them.

Lemma 6.13. *Let T^v be a vertex gadget. Each vertex in T^v is contained in exactly ℓ triangles in $G'[T^v]$.*

Proof. We make a case distinction, that is, for each vertex $a \in T^v$ we present exactly ℓ triangles containing vertex a. We distinguish between the different vertices of a vertex gadget.

Case 1: Consider vertex p^v_d for some $d \in [\ell - 1]$. Because of Observation 6.12 we only have to consider the neighbors of p^v_d in T^v. By construction we have $N(p^v_d) \cap T^v = \{q^v_d\} \cup \{x^v_{1,j} \mid j \in [\ell]\}$. Observe that the only edges within $N(p^v_d) \cap T^v$ are the edges $q^v_d x^v_{1,j}$ for each $j \in [\ell]$. Thus, p^v_d is contained in the ℓ triangles $\{\{p^v_d, q^v_d, x^v_{1,i}\} \mid i \in [\ell]\}$. By similar arguments the same is true for vertex q^v_d.

Case 2: Consider vertex p^v_ℓ. By construction we have $N(p^v_\ell) = \{q^v_\ell\} \cup \{x^v_{s^*,i} \mid i \in [\ell]\} \cup \{q^w_\ell \mid vw \in E(G)\}$ if s is odd or s is even and $\ell \geq 3$. If s is even and $\ell = 2$ we have $N(p^v_\ell) = \{q^v_\ell\} \cup \{x^v_{s^*,i} \mid i \in [\ell]\}$. Observe that the only edges within $N(p^v_\ell)$ are the edges $q^v_\ell x^v_{s^*,i}$ for each $i \in [\ell]$. Thus, p^v_ℓ is contained in the ℓ triangles $\{\{p^v_\ell, q^v_\ell, x^v_{s^*,i}\} \mid i \in [\ell]\}$. By similar arguments the same is true for vertex q^v_ℓ.

Case 3: Now, consider vertices $x^v_{1,i}$ and $x^v_{s^*,i}$ for some $i \in [\ell]$. Here we have to distinguish if $s \in \{3, 4\}$ or $s \geq 5$ since $y^v_{t,i}$ exists only in the second case.

First, consider the case $s = 3$ or $s = 4$. Note that we now have $x^v_{1,i} = x^v_{s^*,i}$. By construction we have $N(x^v_{1,i}) = \{p^v_j \mid j \in [\ell]\} \cup \{q^v_j \mid j \in [\ell]\}$. Note that the only edges within $N(x^v_{1,i})$ have the form $p^v_j q^v_j$ for each $j \in [\ell]$. Thus, $x^v_{1,i}$ is contained in the ℓ triangles $\{\{x^v_{1,i}, p^v_j, q^v_j\} \mid j \in [\ell]\}$.

Second, consider the case $s \geq 5$. First, we investigate vertex $x^v_{1,i}$. By construction we have $N(x^v_{1,i}) = \{p^v_j \mid j \in [\ell - 1]\} \cup \{q^v_j \mid j \in [\ell - 1]\} \cup \{y^v_{1,i}, z^v_{1,i}\}$. The only edges within $N(x^v_{1,i})$ are the edge $p^v_j q^v_j$ for each $j \in [\ell - 1]$ and the edge $y^v_{1,i} z^v_{1,i}$. Thus, $x^v_{1,i}$ is contained in the $\ell - 1$ triangles $\{\{x^v_{1,i}, p^v_j, q^v_j\} \mid j \in [\ell - 1]\}$, and in the triangle $\{x^v_{1,i}, y^v_{1,i}, z^v_{1,i}\}$.

Now, consider vertex $x^v_{s^*,i}$. Because of Observation 6.12 we only have to consider the neighbors of $x^v_{s^*,i}$ in T^v. By construction we observe that $N(x^v_{s^*,i}) \cap T^v = \{p^v_\ell, q^v_\ell\} \cup \{y^v_{s^*-1,j}, z^v_{s^*-1,j} \mid j \in [\ell] \setminus \{i\}\}$. Note that the only edges within $N(x^v_{s^*,i}) \cap T^v$ are the edge $p^v_\ell q^v_\ell$ and the edge $y^v_{s^*-1,j} z^v_{s^*-1,j}$ for each $j \in [\ell] \setminus \{i\}$. Thus, $x^v_{s^*,i}$ is contained in the triangle $\{x^v_{s^*,i}, p^v_\ell, q^v_\ell\}$ and in the $\ell - 1$ triangles $\{\{x^v_{s^*,i}, y^v_{s^*-1,j}, z^v_{s^*-1,j}\} \mid j \in [\ell] \setminus \{i\}\}$.

Case 4: Consider vertex $x^v_{r,i}$ for some $i \in [\ell]$ and some $r \in [2, s^* - 1]$. Recall that these vertices only exist if $s \geq 7$. By construction $N(x^v_{r,i}) = \{y^v_{r,i}, z^v_{r,i}\} \cup \{y^v_{r-1,j}, z^v_{r-1,j} \mid j \in [\ell] \setminus \{i\}\}$. The only edges within $N(x^v_{r,i})$ have the form $y^v_{r,i} z^v_{r,i}$ and $y^v_{r-1,j} z^v_{r-1,j}$ for each $j \in [\ell] \setminus \{i\}$. Thus, $x^v_{r,i}$ is contained in the triangle $\{x^v_{r,i}, y^v_{r,i}, z^v_{r,i}\}$ and in the $\ell - 1$ triangles $\{\{x^v_{r,i}, y^v_{r-1,j}, z^v_{r-1,j}\} \mid j \in [\ell] \setminus \{i\}\}$.

Case 5: Finally, consider vertex $y^v_{t,i}$ for some $i \in [\ell]$ and some $t \in [s^* - 1]$. Recall that these vertices only exist if $s \geq 5$. By construction we have $N(y^v_{t,i}) = \{x^v_{t,i}, z^v_{t,i}\} \cup \{x^v_{t+1,j} \mid j \in [\ell] \setminus \{i\}\}$. The only edges within $N(y^v_{t,i})$ are the edge $x^v_{t,i} z^v_{t,i}$ and the

edge $x^v_{t+1,j}z^v_{t,i}$ for each $j \in [\ell]\setminus\{i\}$. Thus, $y^v_{t,i}$ is contained in the triangle $\{y^v_{t,i}, z^v_{t,i}, x^v_{t,i}\}$ and in the $\ell-1$ triangles $\{\{y^v_{t,i}, z^v_{t,i}, x^v_{t+1,j}\} \mid j \in [\ell]\setminus\{i\}\}$. By similar arguments the same holds for vertex $z^v_{t,i}$.

Thus, each vertex in T^v is contained in exactly ℓ triangles. □

From Lemma 6.13 and Observation 6.12, we conclude the following.

Observation 6.14. *Let S be a vertex-ℓ-triangle s-club for $s = 3$ and $\ell \geq 1$ or for $s \geq 4$ and $\ell \geq 2$ in G'. Then, $S \cap T^v \neq \emptyset \Leftrightarrow T^v \subseteq S$.*

Now, we prove the correctness of Construction 6.11.

Lemma 6.15. *For $s = 3$ and each $\ell \geq 1$, and also for each $s \geq 4$ and $\ell \geq 2$ the* Vertex Triangle s-Club *problem parameterized by k is W[1]-hard, even if each vertex in the input graph is contained in exactly ℓ triangles.*

Proof. We show that G contains a clique of size at least k if and only if G' contains a vertex-ℓ-triangle s-club of size at least $3\ell k s^*$.

Let K be a clique of size at least k in G. We argue that $S \coloneqq \bigcup_{v\in K} T^v$ is a vertex-ℓ-triangle s-club of size at least $3\ell k s^*$ in G'. The size bound follows from the fact that each T^v consists of exactly $3\ell s^*$ vertices. Furthermore, by Lemma 6.13 each vertex in T^v for some $v \in V(G)$ is contained in exactly ℓ triangles in $G'[T^v]$. Hence, it remains to show that S is an s-club.

To do so, we first prove the following two claims. To formulate the claims, we need some further notation. We define $T^v_0 \coloneqq \{p^v_1, \ldots, p^v_{\ell-1}\} \cup \{q^v_1, \ldots, q^v_{\ell-1}\}$ and $T^v_\ell \coloneqq \{p^v_\ell, q^v_\ell\}$. Recall that if $\ell = 1$, then $T^v_0 = \emptyset$. Otherwise, both sets are non-empty.

Claim 3. For $\ell \geq 2$, we have $\mathrm{dist}_{G'}(u,a) + \mathrm{dist}_{G'}(u,b) \leq 2s^*$ for each vertex $a \in T^v_0$, each vertex $b \in T^v_\ell$, and for each vertex $u \in T^v \setminus (T^v_0 \cup T^v_\ell)$.

Proof of Claim. By $X^v_1 \coloneqq \{x^v_{1,j} \mid j \in [\ell]\}$ we denote the neighbors of T^v_0. We define the sets $X^v_2, \ldots, X^v_{s^*}$ and $Y^v_1, Z^v_1, \ldots, Y^v_{s^*-1}, Z^v_{s^*-1}$ analogously. We first make some observations about the neighborhoods of these vertex sets, which then allows us to obtain an upper bound for $\mathrm{dist}_{G'}(u,a) + \mathrm{dist}_{G'}(u,b)$. Note that here we only consider paths that are entirely contained in the vertex gadget T^v.

- For $s \in \{3,4\}$: $N(x^v_{1,i}) \cap T^v = T^v_0 \cup T^v_\ell$ for each $i \in [\ell]$. Hence $N(X^v_1) \cap T^v = T^v_0 \cup T^v_\ell$.
- For $s \geq 5$: $N(x^v_{1,i}) = T^v_0 \cup \{y^v_{1,i}, z^v_{1,i}\}$ for each $i \in [\ell]$. Hence $N(X^v_1) = T^v_0 \cup Y^v_1 \cup Z^v_1$.

- For $s \geq 5$: $N(x^v_{s^*,i}) \cap T^v = T^v_\ell \cup \{y^v_{s^*-1,j}, z^v_{s^*-1,j} \mid j \in [\ell] \setminus \{i\}\}$ for each $i \in [\ell]$. Hence, $N(X^v_{s^*}) \cap T^v = T^v_\ell \cup Y^v_{s^*-1} \cup Z^v_{s^*-1}$.
- For $s \geq 7$: $N(x^v_{t,i}) = \{y^v_{t,i}, z^v_{t,i}\} \cup \{y^v_{t-1,j}, z^v_{t-1,j} \mid j \in [\ell] \setminus \{i\}\}$ for each $t \in [2, s^*-1]$ and each $i \in [\ell]$. Hence, $N(X^v_t) = Y^v_t \cup Z^v_t \cup Y^v_{t-1} \cup Z^v_{t-1}$ for each $t \in [2, s^*-1]$.
- For $s \geq 5$: $N(y^v_{t,i}) = \{x^v_{t,i}, z^v_{t,i}\} \cup \{x^v_{t+1,j} \mid j \in [\ell] \setminus \{i\}\}$ and also $N(z^v_{t,i}) = \{x^v_{t,i}, y^v_{t,i}\} \cup \{x^v_{t+1,j} \mid j \in [\ell] \setminus \{i\}\}$ for each $t \in [s^*-1]$ and each $i \in [\ell]$. Hence, $N(Y^v_t) = X^v_t \cup Z^v_t \cup X^v_{t+1}$ and $N(Z^v_t) = X^v_t \cup Y^v_t \cup X^v_{t+1}$ for each $t \in [s^*-1]$.

From the above and for a vertex $u \in X^v_z$ we obtain a path from u to a of length $2z-1$ by using subsequent vertices from $Y^v_{z-1}, X^v_{z-1}, \ldots, Y^v_1, X^v_1$. A similar observation can be made for a path from u to b. The cases $u \in Y^v_z \cup Z^v_z$ are treated similarly. This implies that $\text{dist}_{G'}(u,a) + \text{dist}_{G'}(u,b) \leq 2s^*$ for each vertex $a \in T^v_0$, each vertex $b \in T^v_\ell$, and for each vertex $u \in T^v \setminus (T^v_0 \cup T^v_\ell)$. ■

Now, we use Claim 3 to show that T^v is a $(2s^*)$-club.

Claim 4. If s is odd, then T^v is an $(s-1)$-club and if s is even, then T^v is an $(s-2)$-club for each vertex $v \in V(G)$.

Proof of Claim. Recall that for $s \in \{3,4\}$ the gadget T^v is a non-induced biclique. Hence, the statement is true in this case. In the following, we assume that $s \geq 5$. Note that this implies that $\ell \geq 2$. We only consider the case that s is odd. The case that s is even follows analogously.

By construction we have $T^v_0 \subseteq N(x^v_{1,1})$ and $T^v_\ell \subseteq N(x^v_{s^*,1})$, so each pair of vertices in T^v_0 has distance at most 2 to each other and the same is true for the vertices in T^v_ℓ.

Consider a pair of vertices $u, \widetilde{u}$ with $u \in T^v_0$ and $\widetilde{u} \in T^v \setminus T^v_0$. From Claim 3 we get $\text{dist}_{G'}(u, \widetilde{u}) \leq s-1$. The case $u \in T^v_\ell$ and $\widetilde{u} \in T^v \setminus T^v_\ell$ follows analogously.

Now, consider a pair of vertices $u, \widetilde{u}$ of $T^v \setminus (T^v_0 \cup T^v_\ell)$. To bound the distance of u and $\widetilde{u}$, we consider one path via a vertex in T^v_0 and one via a vertex in T^v_ℓ. We have $\text{dist}_{G'}(u, \widetilde{u}) \leq \min(\text{dist}_{G'}(u,a) + \text{dist}_{G'}(a, \widetilde{u}), \text{dist}_{G'}(u,b) + \text{dist}_{G'}(b, \widetilde{u}))$ for each vertex $a \in T^v_0$, and each vertex $b \in T^v_\ell$. From Claim 3 we know that $\text{dist}_{G'}(u,a) + \text{dist}_{G'}(u,b) \leq 2s^*$ and that $\text{dist}_{G'}(\widetilde{u}, a) + \text{dist}_{G'}(\widetilde{u}, b) \leq 2s^*$. Hence, $\text{dist}_{G'}(u, \widetilde{u}) \leq 2s^* \leq s-1$. Thus, T^v is indeed an $(s-1)$-club. ■

Now, we show that for two vertices v and w in K, a vertex $u \in T^v$, and a vertex $\widetilde{u} \in T^w$ we have $\text{dist}_{G'}(u, \widetilde{u}) \leq s$. We consider the three cases:

Case I: s is odd. Observe that since $vw \in E(G)$, each vertex $u \in (T^v_0 \cup T^v_\ell)$ has one neighbor in T^w. Since T^w is an $(s-1)$-club by Claim 4, we obtain that for each vertex $\widetilde{u} \in T^w$ we have $\text{dist}_{G'}(u, \widetilde{u}) \leq s$ if $u \in T^v_0 \cup T^v_\ell$. Hence, it remains to consider the case that $u \in T^v \setminus (T^v_0 \cup T^v_\ell)$ and that $\widetilde{u} \in T^w \setminus (T^w_0 \cup T^w_\ell)$. For this,

let $u_1^v \coloneqq \text{dist}_{G'}(u, p_1^v), \widetilde{u}_1^w \coloneqq \text{dist}_{G'}(q_1^w, \widetilde{u}), u_\ell^v \coloneqq \text{dist}_{G'}(u, p_\ell^v)$, and $\widetilde{u}_\ell^w \coloneqq \text{dist}_{G'}(q_\ell^w, \widetilde{u})$. Note that

$$\begin{aligned}\text{dist}_{G'}(u, \widetilde{u}) &\leq \min(u_1^v + 1 + \widetilde{u}_1^w, u_\ell^v + 1 + \widetilde{u}_\ell^w) \\ &= 1 + \min(u_1^v + \widetilde{u}_1^w, u_\ell^v + \widetilde{u}_\ell^w).\end{aligned}$$

In this inequality, the '+1' is the result of the fact that we have to use an edge to get from gadget T^v to gadget T^w. By Claim 3 we know that $u_1^v + u_\ell^v \leq 2s^*$ and that $\widetilde{u}_1^w + \widetilde{u}_\ell^w \leq 2s^*$. Since s is odd, we obtain that $\text{dist}_{G'}(u, \widetilde{u}) \leq 1 + 2s^* = s$. Thus, S is a vertex-ℓ-triangle s-club of size $3\ell k s^*$.

Case II: s is even and $\ell \geq 3$. For each vertex pair $u \in T^v \setminus T_0^v$ and $\widetilde{u} \in T^w \setminus T_0^w$ the proof of $\text{dist}_{G'}(u, \widetilde{u}) \leq s$ is analogous to the proof in Case I handling odd values of s. Hence, it remains to show that each vertex $u \in T_0^v$ has distance at most s to each vertex $\widetilde{u} \in T^w$. Observe that $\text{dist}_{G'}(u, p_1^v) \leq 2$ since $T_0^v \subseteq N(x_{1,1}^v)$ and $x_{1,1}^v$ is a neighbor of p_1^v. Thus, $\text{dist}_{G'}(u, q_1^w) \leq 3$ since $vw \in E(G)$. Furthermore, for each vertex $\widetilde{u} \in T^w \setminus T_\ell^w$ we have $\text{dist}_{G'}(\widetilde{u}, q_1^w) \leq s - 3$ by the proof of Claim 4. Hence, $\text{dist}(u, \widetilde{u}) \leq s$. Thus, it remains to consider the case that $u \in T_0^v$ and that $\widetilde{u} \in T_\ell^w = \{p_\ell^w, q_\ell^w\}$. Since $vw \in E(G)$ we conclude that $\widetilde{u}$ has a neighbor in T^v. Since T^v is an $(s-2)$-club by Claim 4 we conclude that $\text{dist}_{G'}(u, \widetilde{u}) = s - 1$. Hence, S is a vertex-ℓ-triangle s-club of size $3\ell k s^*$.

Case III: s is even and $\ell = 2$. Let $x^w \coloneqq x_{s^*,1}^w$ and $x^v \coloneqq x_{s^*,1}^v$. Furthermore, let $u_1^v \coloneqq \text{dist}_{G'}(u, p_1^v), \widetilde{u}_x^w \coloneqq \text{dist}_{G'}(x^w, \widetilde{u}), u_x^v \coloneqq \text{dist}_{G'}(u, x^v)$, and $\widetilde{u}_1^w \coloneqq \text{dist}_{G'}(p_1^w, \widetilde{u})$. We have

$$\begin{aligned}\text{dist}_{G'}(u, \widetilde{u}) &\leq \min(u_1^v + 1 + \widetilde{u}_x^w, u_x^v + 1 + \widetilde{u}_1^w) \\ &= 1 + \min(u_1^v + \widetilde{u}_x^w, u_x^v + \widetilde{u}_1^w).\end{aligned}$$

Again, the '+1' results from the fact that we have to use an edge to go from gadget T^v to T^w. Consider the following claim.

Claim 5. For each vertex $u \in T^v$ we have $u_1^v + u_x^v \leq s - 1$.

Claim 5 directly implies that

$$\min(u_1^v + \widetilde{u}_x^w, u_x^v + \widetilde{u}_1^w) \leq s - 1$$

and thus $\text{dist}_{G'}(u, \widetilde{u}) \leq s$. This in turn implies that S is a vertex-ℓ-triangle s-club of size $3\ell k s^*$. Thus, it remains to prove Claim 5.

Proof of Claim. We first consider the case that $u \in T_\ell^v$. By construction we have $T_\ell^v \subseteq N(x_{s^*,1}^v)$ and thus $u_x^v = 1$. Since T^v is an $(s-2)$-club by Claim 4, we have $u_1^v \leq s - 2$ and hence Claim 5 is true in this case.

Next, we consider the case that $u \in T_0^v$. Since $T_0^v \subseteq N(x_{1,1}^v)$ we conclude that $u_1^v \leq 2$. Furthermore, $T_0^v \subseteq N(x_{1,i}^v)$ for each $i \in [\ell]$, so there is a path of length $2(s^*-1)+1 = s-3$ from u to $x_{s^*,1}^v$ and we conclude that $u_x^v = s-3$. Hence, Claim 5 is true in this case.

For all remaining cases, it is sufficient to prove the claim for each vertex $u \in T^v \setminus (T_0^v \cup T_\ell^v)$. First, we consider the case that $u \coloneqq x_{t,i}^v$ for some $t \in [s^*]$ and some $i \in [\ell]$. Then we have $\text{dist}_{G'}(x_{t,i}^v, p_1^v) = 2t-1$ and $\text{dist}_{G'}(x_{t,i}^v, x_{s^*,1}^v) \leq 2(s^*-t)+2$ and hence $\text{dist}_{G'}(x_{t,i}^v, p_1^v) + \text{dist}_{G'}(x_{t,i}^v, x_{s^*,1}^v) \leq 2t-1+2s^*-2t+2 = 2s^*+1 \leq s-1$.

Second, we consider the case that $u \coloneqq y_{t,i}^v$ for some $t \in [s^*-1]$ and some $i \in [\ell]$. Then we have $\text{dist}_{G'}(y_{t,i}^v, p_1^v) = 2t$ and $\text{dist}_{G'}(y_{t,i}^v, x_{s^*,1}^v) \leq 2(s^*-t)+1$ and hence we obtain that $\text{dist}_{G'}(y_{t,i}^v, p_1^v) + \text{dist}_{G'}(y_{t,i}^v, x_{s^*,1}^v) \leq 2t+2s^*-2t+1 = 2s^*+1 \leq s-1$. The case that $u \coloneqq z_{t,i}^v$ follows by the same argumentation. ■

Conversely, suppose that G' contains a vertex-ℓ-triangle s-club S of size at least $3\ell k s^*$. Because of Observation 6.14 for each vertex gadget T^v we either have $T^v \subseteq S$ or $T^v \cap S = \emptyset$. Hence, S contains at least k vertex gadgets. We assume towards a contradiction that S contains two vertex gadgets T^v and T^w such that $vw \notin E(G)$. In each case, we will determine a vertex $u_v \in T^v$ and a vertex $u_w \in T^w$ such that $\text{dist}_{G'}(u_v, u_w) \geq s+1$. This contradiction to the s-club property allows us to conclude that the set $\{v \mid T^v \subseteq S\}$ is a clique of size at least k in G.

Case I: s is odd. We define vertex u_v as follows. The vertex u_w is defined analogously. Recall that in this case we have $s^* = (s-1)/2$.

- If $s \equiv 3 \mod 4$, we set $u_v \coloneqq x_{(s+1)/4,1}^v$.
- Otherwise, if $s \equiv 1 \mod 4$, we set $u_v \coloneqq y_{(s-1)/4,1}^v$.

Observe that for each vertex $u \in T_0^v \cup T_\ell^v$ we have $\text{dist}_{G'}(u_v, u) = (s-1)/2$. Furthermore, recall that the vertices in $T_0^v \cup T_\ell^v$ are the only vertices in T^v with neighbors in other vertex gadgets.

Similarly, for each vertex $u' \in T_0^w \cup T_\ell^w$ we have $\text{dist}_{G'}(u_w, u') = (s-1)/2$. But since $vw \notin E(G)$ there are no edges between T^v and T^w and we obtain for each choice of u and u' that $\text{dist}_{G'}(u_v, u_w) \geq \text{dist}_{G'}(u_v, u) + 2 + \text{dist}_{G'}(u', u_w) = (s-1)/2 + 2 + (s-1)/2 = s+1$, a contradiction.

Case II: s is even and $\ell \geq 3$. We set $u_v \coloneqq p_2^v$ and $u_w \coloneqq x_{s^*,1}^w$. Note that $u_v \in T_0^v$ since $\ell \geq 3$. By the construction we obtain that for each vertex $u_0 \in \{p_1^v, q_1^v\}$ we have $\text{dist}_{G'}(u_v, u_0) = 2$ since $T_0^v \subseteq N(x_{1,1}^v)$. Furthermore, for each vertex $u_\ell \in \{p_\ell^v, q_\ell^v\}$ we have $\text{dist}_{G'}(u_v, u_\ell) = s-2$. Next, observe that for each vertex $u_0' \in \{p_1^w, q_1^w\}$ we have $\text{dist}_{G'}(u_w, u_0') = s-3$ and for each vertex $u_\ell' \in \{p_\ell^w, q_\ell^w\}$ we have $\text{dist}(u_w, u_\ell') = 1$.

Since $vw \notin E(G)$, there are no edges between the gadgets T^v and T^w. Hence,

$$\begin{aligned}\text{dist}_{G'}(u_v, u_w) \geq \min(&\text{dist}_{G'}(u_v, u_0) + 2 + \text{dist}_{G'}(u'_0, u_w),\\ &\text{dist}_{G'}(u_v, u_\ell) + 2 + \text{dist}_{G'}(u'_\ell, u_w))\end{aligned}$$

for each vertex $u_0 \in \{p_1^v, q_1^v\}$, each $u'_0 \in \{p_1^w, q_1^w\}$, each $u_\ell \in T_\ell^v$, and each $u'_\ell \in T_\ell^w$. By the above argumentation we obtain $\text{dist}_{G'}(u_v, u_w) \geq \min(2+2+s-3, s-2+2+1) = s+1$, a contradiction.

Case III: s is even and $\ell = 2$. We define the vertices u_v and u_w as follows:

- If $s = 4$, we set $u_v \coloneqq x_{1,2}^v$ and $u_w \coloneqq p_2^w$.
- If $s \equiv 0 \mod 4$ and $s \geq 8$, we set $u_v \coloneqq x_{s/4,1}^v$ and $u_w \coloneqq y_{s/4,1}^w$.
- If $s \equiv 2 \mod 8$, we set $u_v \coloneqq y_{(s-2)/4,2}^v$ and $u_w \coloneqq x_{(s+2)/4,1}^w$.
- If $s \equiv 6 \mod 8$, we set $u_v \coloneqq y_{(s-2)/4,1}^v$ and $u_w \coloneqq x_{(s+2)/4,2}^w$.

From the definition of these vertices we obtain that $\text{dist}_{G'}(u_v, p_1^v) = s/2 - 1$, $\text{dist}_{G'}(u_v, x_{s^*,1}^v) = s/2$, $\text{dist}_{G'}(u_w, p_1^w) = s/2$, and that $\text{dist}_{G'}(u_w, x_{s^*,1}^w) = s/2 - 1$. Recall that the vertices p_1^u and $x_{s^*,1}^u$ are the only vertices in T^u which have neighbors outside T^u for each $u \in V(G)$. Furthermore, observe that all neighbors of p_1^u which are not contained in T^u are the vertices $x_{s^*,1}^b$ where $ub \in E(G)$. Similar, all neighbors of $x_{s^*,1}^u$ which are not in T^u are of the form p_1^b where $ub \in E(G)$. We conclude that $\text{dist}_{G'}(u_v, u_w) \geq \text{dist}_{G'}(u_v, p_1^v) + 3 + \text{dist}_{G'}(x_{s^*,1}^w, u_w)$. Here, the '+3' results from the fact that at least 3 edges to switch the vertex gadgets have to be used: one is not sufficient since $uw \notin E(G)$ and also two are not sufficient since in two steps one can only reach a vertex p_1^c for $c \in V(G)$ from p_1^v but no vertex $x_{s^*,1}^d$ for $d \in V(G)$. Hence, $\text{dist}_{G'}(u_v, u_w) \geq s/2 - 1 + 3 + s/2 - 1 = s + 1$, a contradiction. □

6.2 Edge Triangle s-Club

In this section we settle the parameterized complexity of Edge Triangle s-Club with respect to the solution size k. Recall that a vertex set S is an edge-ℓ-triangle s-club if $G[S]$ contains a spanning subgraph $G' = (S, E')$ such that each edge in $E(G')$ is contained in at least ℓ triangles within G' and the diameter of G' is at most s. First, we show that Edge Triangle s-Club is FPT with respect to k when $\ell = 1$ irrespective of the value of s by providing a Turing kernel. To show this, it is sufficient to delete edges which are not part of a triangle. Afterwards, we prove W[1]-hardness of Edge Triangle s-Club with respect to k for all fixed $\ell \geq 2$.

6.2.1 Edge Triangle s-Club with $\ell = 1$

Now, we prove that Edge Triangle s-Club for $\ell = 1$ admits a Turing kernel with respect to k implying that the problem is FPT. To obtain the kernel we need the following reduction rule which removes edges which are in no triangle.

Reduction Rule 6.2. *Let (G, k) be an instance of* Edge Triangle s-Club. *Delete all edges from G which are not part of any triangle.*

It is clear that Reduction Rule 6.2 is correct and can be applied in polynomial time. The idea is that after Reduction Rule 6.2 is applied, we can bound the size of the neighborhood of each vertex. Next, we prove that after the application of Reduction Rule 6.2 each edge with both endpoints in the closed neighborhood of a vertex w is contained in a triangle which is contained completely in $N[w]$.

Lemma 6.16. *Let (G, k) be an instance of* Edge Triangle s-Club *with $\ell = 1$ to which Reduction Rule 6.2 is applied. For each vertex $v \in V(G)$ each edge in $G[N[v]]$ is contained in at least one triangle in $G[N[v]]$.*

Proof. First, we consider edges of the form uv where u is a neighbor of v. Since Reduction Rule 6.2 is applied, there exists another vertex $w \in V(G)$ such that $G[\{u, v, w\}]$ is a triangle. Observe that $w \in N(v)$.

Second, each edge uw with $u, w \in N(v)$ is in a triangle with vertex v. □

Next, we show that any instance such that sufficiently many vertices are close to some vertex v is a yes-instance.

Lemma 6.17. *Let (G, k) be an instance of* Edge Triangle s-Club *with $\ell = 1$ to which Reduction Rule 6.2 is applied. Then, (G, k) is a yes-instance if $|N_{\lfloor s/2 \rfloor}[v]| \geq k$ for some vertex $v \in V(G)$.*

Proof. By Lemma 6.16 each edge in $G[N_{\lfloor s/2 \rfloor}[v]]$ is contained in at least one triangle in $G[N_{\lfloor s/2 \rfloor}[v]]$ since $N_{\lfloor s/2 \rfloor}[v] = \bigcup_{w \in N_{\lfloor s/2 \rfloor - 1}[v]} N[w]$ for each $s \geq 4$. Furthermore, each vertex in $N_{\lfloor s/2 \rfloor}[v]$ has distance at most $\lfloor s/2 \rfloor$ to vertex v. Hence, $N_{\lfloor s/2 \rfloor}[v]$ is an s-club and by assumption $|N_{\lfloor s/2 \rfloor}[v]| \geq k$. □

Lemma 6.17 implies a Turing kernel for k which implies that the problem is fixed-parameter tractable.

Theorem 6.18. Edge Triangle s-Club *for $\ell = 1$ admits a k^2-vertex Turing kernel if s is even and a k^3-vertex Turing kernel if s is odd and $s \geq 3$.*

Proof. First, we apply Reduction Rule 6.2. Because of Lemma 6.17 we conclude that (G, k) is a trivial yes-instance, if $|N_{\lfloor s/2 \rfloor}[v]| \geq k$ for some $v \in V(G)$. Hence, in the following we can assume that $k > |N_{\lfloor s/2 \rfloor}[v]|$ for each vertex $v \in V(G)$.

First, we consider the case that s is even. Then $\lfloor s/2 \rfloor = s/2$ and we obtain that $N_s[v] \subseteq N_{s/2}[N_{s/2}[v]]$ for each $v \in V(G)$. Thus, $|N_s[v]| \leq k^2$.

Second, we consider the case that s is odd. Observe that we have $N_s[v] \subseteq N_{\lfloor s/2 \rfloor}[N_{\lfloor s/2 \rfloor}[N_{\lfloor s/2 \rfloor}[v]]]$ for each $v \in V(G)$. Thus, $|N_s[v]| \leq k^3$. □

6.2.2 Edge Triangle s-Club for $\ell \geq 2$

Now we show W[1]-hardness for the remaining cases.

Theorem 6.19. Edge Triangle s-Club *is W[1]-hard for parameter k if $\ell \geq 2$.*

Next, we describe the construction of the reduction to prove Theorem 6.19. We reduce from Clique. The idea is to construct one vertex gadget for each vertex of the Clique instance and to add edges between two different vertex gadgets if and only if the two corresponding vertices are adjacent in such a way that all these edges are in exactly ℓ triangles. For an illustration of this construction see Figure 6.5.

Construction 6.20. Let (G, k) be an instance of Clique with $k \geq 3$. We construct an equivalent instance (G', k') of Edge-Triangle s-Club for some fixed $\ell \geq 2$ as follows. Let $\ell^* := \lceil \ell/2 \rceil$ and let $x := 6 \cdot \ell^*(s-1) + \lfloor \ell/2 \rfloor$. For each vertex $v \in V(G)$, we construct the following vertex gadget T^v. For better readability, all sub-indices of the vertices in T^v are considered modulo x. Our construction distinguishes between even and odd values of ℓ. First, we describe the part of the construction which both cases have in common.

1. We add vertex sets $A_v := \{a_i^v \mid i \in [0, x]\}$ and $B_v := \{b_i^v \mid i \in [0, x]\}$ to G'.
2. We add the edges $a_i^v a_{i+j}^v$, and $b_i^v b_{i+j}^v$ for each $i \in [0, x]$ and each $j \in [-3\ell^*, 3\ell^*] \setminus \{0\}$ to G'.
3. We add the edge $a_i^v b_{i+j}^v$ for each $i \in [0, x]$ and each $j \in [-3\ell^*, 3\ell^*]$ to G'.

In other words, an edge $a_i^v b_j^v$ is added if the indices differ by at most $3\ell^*$. For even ℓ, this completes the construction of T^v. For odd ℓ, we extend T^v as follows:

0-1. We add the vertex set $C_v := \{c_i^v \mid i \in [0, x] \text{ and } i \equiv 0 \mod \ell^*\}$ to G'. Note that C_v consists of exactly $6s - 5$ vertices.

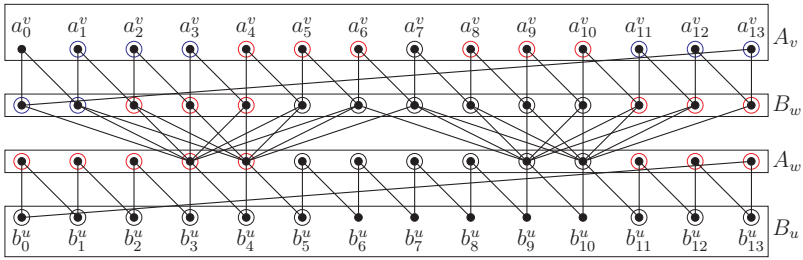

Figure 6.5: Construction for Theorem 6.19 when $s = 3$ and $\ell = 2$ and G is a P_3 on $\{u, v, w\}$ with $uv \notin E(G)$. Only the gadgets A_v, B_w, A_w, and B_u are shown. For simplicity, no edges within A_v, B_w, A_w, and B_u are drawn and edges between B_w and A_w are only drawn if one endpoint is a_3^w, a_4^w, a_9^w, or a_{10}^w.
Blue encircled vertices are neighbors of a_0^v, red encircled vertices have distance 2 to a_0^v, and black encircled vertices have distance 3 to a_0^v. Thus, a_0^v and $b_7^u = b_{0+1+3\cdot1\cdot2}^u = b_{0+\lfloor \ell/2 \rfloor+3\ell^*(s-1)}^u$ have distance at least 4.

0-2. We add the edges $c_i^v a_{i+j}^v$ and $c_i^v b_{i+j}^v$ for each $i \in [0, x]$ such that $i \equiv 0 \mod \ell^*$ and each $j \in [-3\ell^*, 3\ell^*]$ to G'.

0-3. Also, we add the edge $c_i^v c_{i+j}^v$ to G' for each $i \in [0, x]$ such that $i \equiv 0 \mod \ell^*$ and each $j \in [-3\ell^*, 3\ell^*] \setminus \{0\}$ to G' if the corresponding vertex c_{i+j}^v exists.

In other words, an edge between c_i^v and a_j^v, b_j^v, or c_j^v is added if the indices differ by at most $3\ell^*$. Now, for each edge $uv \in E(G)$, we add the following to G':

0-4. We add the edges $a_i^v b_{i+j}^u$ and $a_i^u b_{i+j}^v$ for each $i \in [0, x]$ and $j \in [0, \lfloor \ell/2 \rfloor]$.

0-5. If ℓ is odd, we also add the edges $c_i^v b_{i+j}^u$ and $c_i^u b_{i+j}^v$ for each $i \in [0, x]$ such that $i \equiv 0 \mod \ell^*$ and each $j \in [0, \lfloor \ell/2 \rfloor]$ to G'.

Observe that each vertex b_{i+j}^u is adjacent to *exactly* one vertex in C_v.

In other words, an edge between a_i^v or c_i^v and b_j^v is added if j exceeds i by at most $\lfloor \ell/2 \rfloor$. Finally, if ℓ is even, we set $k' := 2(x+1)k = (\ell(6s-5)+2) \cdot k$, and if ℓ is odd, we set $k' := (2(x+1)+6s-5)k = (\ell+2)(6s-5) \cdot k$.

Construction 6.20 has two key mechanisms: First, if $uv \notin E(G)$ then for each vertex $a \in A_v$ there is at least one vertex $b \in B_u$ such that $\text{dist}(a, b) > s$. Second, each edge with one endpoint in A_v and one endpoint in B_u is contained in *exactly* ℓ

triangles. Furthermore, if ℓ is odd, then this also holds for each edge with one endpoint in C_v and one in B_u. Consider an edge-ℓ-triangle s-club S in G and let $\widetilde{G} = (S, \widetilde{E})$ be a spanning subgraph of $G[S]$ with the maximal number of edges, such that each edge of $\widetilde{E}$ is contained in at least ℓ triangles in $\widetilde{G}$ and the diameter of $\widetilde{G}$ is s. As we will show, the two mechanisms ensure that an edge with one endpoint in A_v (or C_v) and the other endpoint in B_u is contained in $\widetilde{E}$ if and only if S contains all vertices of A_v (and C_v) and B_u. We call this the *enforcement property*. Next, we formalize this property. To this end, we introduce the following notation. By E_{uv} we denote the set of all edges with one endpoint in A_v (or C_v if ℓ is odd) and the other endpoint in B_u.

Lemma 6.21. *Let S be an edge-ℓ-triangle s-club in the graph G' constructed in Construction 6.20. More precisely, let $\widetilde{G} = (S, \widetilde{E})$ be a maximal subgraph of $G'[S]$ such that each edge in $E(\widetilde{G})$ is contained in at least ℓ triangles within $\widetilde{G}$ and the diameter of $\widetilde{G}$ is at most s. Let $e \in E_{uv}$. Then $e \in E(\widetilde{G})$ if and only if $A_v, B_u \subseteq S$ (and $C_v \subseteq S$, if ℓ is odd).*

Proof. Before we show the two implications, we prove the following cascading property of edge-ℓ-triangle s-clubs which contain at least one edge of E_{uv}.

Claim 6. If $a_i^v b_j^u \in E(\widetilde{G})$ or $c_i^v b_j^u \in E(\widetilde{G})$, then $E_{uv} \subseteq E(\widetilde{G})$.

Proof of Claim. First, we consider even values of ℓ. Note that $\ell^* = \lfloor \ell/2 \rfloor = \ell/2 = \lceil \ell/2 \rceil$. By construction we have:

- $N[a_i^v] = \{a_{i+i'}^v, b_{i+i'}^v \mid i' \in [-3\ell/2, 3\ell/2]\} \cup \{b_{i+i'}^w \mid i' \in [0, \ell/2] \text{ and } vw \in E(G)\}$, and
- $N[b_j^u] = \{a_{j+i'}^u, b_{j+i'}^u \mid i' \in [-3\ell/2, 3\ell/2]\} \cup \{a_{j-i'}^w \mid i' \in [0, \ell/2] \text{ and } uw \in E(G)\}$.

Since $a_i^v b_j^u \in E(G')$ we obtain by Part 0-4. of Construction 6.20 that $j = i + z$ for some $z \in [0, \ell/2]$. Without loss of generality we assume that $z > 0$. Let $y \coloneqq \ell/2 - z$ and observe that $N(a_i^v) \cap N(b_j^u) = \{a_{i+i'}^v \mid i' \in [-y, z] \setminus \{0\}\} \cup \{b_{j+i'}^u \mid i' \in [-z, y] \setminus \{0\}\}$. Thus, the edge $a_i^v b_j^u$ is contained in exactly ℓ triangles whose vertex sets are all contained in $G'[A_v \cup B_u]$. Since $a_i^v b_j^u \in E(\widetilde{G})$, we thus conclude that all vertices and edges which form these ℓ triangles are contained in $\widetilde{G}$. Since $a_i^v b_j^u \in E(\widetilde{G})$ and we assume that $z > 0$, we obtain that $a_{i+1}^v \in V(\widetilde{G})$ and $a_{i+1}^v b_j^u \in E(\widetilde{G})$. Now, $j = i+1+z'$ where $z' = z - 1$ and thus $z < \ell/2$. By similar arguments as before, we can show that $a_{i+1}^v b_{j+1}^u \in E(\widetilde{G})$ and thus also that $a_{i+1}^v, b_{j+1}^u \in V(\widetilde{G})$. Now, since $a_{i+1}^v b_{j+1}^u \in E(\widetilde{G})$ we can repeat the above argumentation for the edge $a_{i+1}^v b_{j+1}^u$ and inductively

for the edge $a^v_{i+q}b^v_{i+q}$ for all $q \in [x]$. We then have verified that $A_v \cup B_u \subseteq V(\widetilde{G})$ and that each edge in E_{uv} is contained in $E(\widetilde{G})$. Recall that A_v and B_u have size $x+1$.

Second, we consider odd values of ℓ, that is $\ell = 2t+1$ for some integer t. Note that $\lfloor \ell/2 \rfloor = t$ and that $\ell^* = \lceil \ell/2 \rceil = t+1$. Furthermore, observe that vertex b^u_j has exactly one neighbor $c^v_{j-i'}$ in C_v for some $i' \in [0,t]$ such that $j - i' \mod (t+1) = 0$.

Hence, by construction we have

- $N[a^v_i] = N[c^v_i] = \{a^v_{i+i'}, b^v_{i+i'} \mid i' \in [-3(t+1), 3(t+1)]\} \cup \{c^v_{i+i'} \mid i' \in [-3(t+1), 3(t+1)]$ and $(i+i') \mod (t+1) = 0\} \cup \{b^w_{i+i'} \mid i' \in [0,t]$ and $vw \in E(G)\}$, and
- $N[b^u_j] = \{a^u_{j+i'}, b^u_{j+i'} \mid i' \in [-3(t+1), 3(t+1)]\} \cup \{c^u_{j+i'} \mid i' \in [-3(t+1), 3(t+1)]$ and $(j+i') \mod (t+1) = 0\} \cup \{a^w_{j-i'} \mid i' \in [0,t]$ and $uw \in E(G)\} \cup \{c^w_{j-i'} \mid i' \in [0,t]$ and $(j-i') \mod (t+1) = 0$ and $uw \in E(G)\}$.

Since $a^v_i b^u_j \in E(G')$ we obtain by Part 0-4. of Construction 6.20 that $j = i + z$ for some $z \in [0,t]$. Without loss of generality we assume that $z > 0$. Now, let $y \coloneqq t - z$ and let $c^v_{i'}$ be the unique neighbor of b^u_j in C_v. We conclude that $N(a^v_i) \cap N(b^u_j) = \{a^v_{i+j'} \mid j' \in [-y, z] \setminus \{0\}\} \cup \{b^u_{j+j'} \mid j' \in [-z, y] \setminus 0\} \cup \{c^v_{i'}\}$.

Hence, both vertices have exactly $t+t+1 = \ell$ common neighbors and thus $N(a^v_i) \cap N(b^u_j) \subseteq S$. By similar arguments a similar statement can be shown for the edge $c^v_i b^u_j$. Thus, the edges $a^v_i b^u_j$ and $c^v_i b^u_j$ are contained in exactly ℓ triangles whose vertex sets are all contained in $G'[A_v \cup C_v \cup B_u]$. Since $a^v_i b^u_j \in E(\widetilde{G})$ we thus conclude that all vertices and edges which form these ℓ triangles are contained in $\widetilde{G}$.

Furthermore, since $a^v_i b^u_j \in E(\widetilde{G})$ and we assume that $z > 0$, we obtain that $a^v_{i+1} \in V(\widetilde{G})$ and $a^v_{i+1} b^u_j \in E(\widetilde{G})$. Now, $j = i + 1 + z'$ where $z' = z - 1$ and thus $z < \ell/2$. By similar arguments as before, we can show that $a^v_{i+1} b^u_{j+1} \in E(\widetilde{G})$ and thus also that $a^v_{i+1}, b^u_{j+1} \in V(\widetilde{G})$. Now, since $a^v_{i+1} b^u_{j+1} \in E(\widetilde{G})$ we can repeat the above argumentation for the edge $a^v_{i+1} b^u_{j+1}$ and inductively for the edge $a^v_{i+q} b^v_{i+q}$ for all $q \in [x]$. We then have verified that $A_v \cup B_u \cup C_v \subseteq V(\widetilde{G})$ and that each edge in E_{uv} is contained in $E(\widetilde{G})$. ■

Now, we are ready to prove the two implications. The implication that if $e \in E(\widetilde{G})$ then $A_v, B_u \subseteq S$ (and $C_v \subseteq S$, if ℓ is odd) directly follows from Claim 6. It remains to show the other implication. From Claim 6 we conclude that either $E_{uv} \subseteq E(\widetilde{G})$ or $E_{uv} \cap E(\widetilde{G}) = \emptyset$. If $E_{uv} \subseteq E(\widetilde{G})$, then we are done, so we assume towards a contradiction that $E_{uv} \cap E(\widetilde{G}) = \emptyset$. Recall that $A_v \cup B_u \subseteq S$ (and also $C_v \subseteq S$ is ℓ is odd). Furthermore, recall that $\widetilde{G}$ is maximal, that is, there exists no spanning subgraph of $G'[S]$ which has more edges than $\widetilde{G}$. The proof of Claim 6 also shows

that each edge in E_{uv} is contained in exactly ℓ triangles. Thus, adding all edges in E_{uv} to $\widetilde{G}$ is still an edge-ℓ-triangle s-club, a contradiction to the assumption that $\widetilde{G}$ contains the maximal number of possible edges. □

Now, we prove the correctness of the reduction for Theorem 6.19.

Proof of Theorem 6.19. We show that G contains a clique of size at least k if and only if G' contains an edge-ℓ-triangle s-club of size at least k'.

Let K be a clique of size k in G. Recall that T^v is the gadget of vertex $v \in V(G)$. We verify that $S \coloneqq \{v \in V(T^v) \mid v \in K\}$ is an edge-ℓ-triangle s-club of size at least k'. More precisely, we show that S fulfills all properties of being an edge-ℓ-triangle s-club. Since $|T^v| = 2\ell^*(6s-5)+2$ if ℓ is even and $|T^v| = (2\ell^*+1)(6s-5)$ if ℓ is odd for each $v \in K$ and since $|K| \geq k$, we have $|S| = |V(\widetilde{G})| \geq k'$. It remains to show that S is an edge-ℓ-triangle s-club. First, we prove the s-club property.

Claim 7. $\widetilde{G}$ is an s-club.

Proof of Claim. First, we show that T^v is an s-club. For this, consider the vertex pair $\{a_i^v, a_j^v\}$ for some $v \in K$. Observe that $P \coloneqq (a_i^v, a_{i+1}^v, \ldots, a_{i+p}^v)$ for $i+p=j$ is a path of length p from a_i^v to a_j^v and that $Q \coloneqq (a_i^v, a_{i-1}^v, \ldots, a_{i-q}^v)$ for $i-q=j$ is a path of length q from a_i^v to a_j^v. Clearly, $p+q = x+1$. Hence, $\min(p,q) \leq (x+1)/2 \leq 3\ell^*(s-1) + \lfloor \ell/2 \rfloor$. Without loss of generality, assume that the minimum is achieved by path P and assume that $p = \alpha \cdot (3\ell^*) + \beta$ for some $\alpha \in [s-1]$ and some $\beta < 3\ell^*$. Recall that by Part 2 of Construction 6.20, $a_{i'}^v a_{j'}^v \in E(G')$ if and only if $j' = i' + z$ for some $z \in [-3\ell^*, 3\ell^*] \setminus \{0\}$. Hence, $(a_i^v, a_{i+1\cdot(3\ell^*)}^v, \ldots, a_{i+\alpha\cdot(3\ell^*)}^v, a_{i+\alpha\cdot(3\ell^*)+\beta}^v)$ is a path of length at most $(s-1)+1 = s$ from a_i^v to a_j^v.

These arguments also apply symmetrically to the vertex pairs $\{b_i^v, b_j^v\}$ and $\{a_i^v, b_j^v\}$ for each $v \in K$. Furthermore, if ℓ is odd, observe that the above argumentation can also be used to show that the vertex pairs $\{c_i^v, a_j^v\}$, $\{c_i^v, b_j^v\}$, and $\{c_i^v, c_j^v\}$ have distance at most s to each other.

Second, we show that a_i^v has distance at most s to b_j^u. Note that by Part 0-4. of Construction 6.20, a_i^v has neighbors $b_i^u, \ldots, b_{i+\lfloor \ell/2 \rfloor}^u$ since $uv \in E(G)$. In the following, we assume that $j \neq i+z$ for all $z \in [0, \lfloor \ell/2 \rfloor]$. Consider the paths $P \coloneqq (a_i^v, b_{i+\lfloor \ell/2 \rfloor}^u, b_{i+\lfloor \ell/2 \rfloor+1}^u, \ldots, b_{i+\lfloor \ell/2 \rfloor+p}^u)$ for $i + \lfloor \ell/2 \rfloor + p = j$ of length $p+1$ and $Q \coloneqq (a_i^v, b_i^u, b_{i-1}^u, \ldots, b_{i-q}^u)$ for $i-q=j$ of length $q+1$. Observe that $(p+1)+(q+1) = (x+3) - \lfloor \ell/2 \rfloor$. Thus, $p+q = (x+1) - \lfloor \ell/2 \rfloor = 6\ell^*(s-1)+1$. Since p and q are integers we have $\min(p,q) \leq \lfloor ((x+1) - \lfloor \ell/2 \rfloor)/2 \rfloor = 3\ell^*(s-1)$.

Without loss of generality assume that the minimum is achieved by path P and assume that $p = \alpha \cdot (3\ell^*) + \beta$ for some $\alpha \in [s-2]$ and some $\beta \in [3\ell^*]$. Recall that

by Part 2 of Construction 6.20, we have $b_i^u b_{j'}^u \in E(G')$ if and only if $j' = i' + z$ for some $z \in [-3\ell^*, 3\ell^*] \setminus \{0\}$. Now, observe that

$$(a_i^v, b^u_{i+\lfloor \ell/2 \rfloor}, b^u_{i+\lfloor \ell/2 \rfloor+1\cdot(3\ell^*)}, \ldots, b^u_{i+\lfloor \ell/2 \rfloor+\alpha\cdot(3\ell^*)}, b^u_{i+\lfloor \ell/2 \rfloor+\alpha\cdot(3\ell^*)+\beta})$$

is a path of length at most $1 + (s-2) + 1 = s$ from a_i^v to b_j^u.

Furthermore, if ℓ is odd, observe that the above argumentation can also be used to show that the vertex pairs $\{c_i^v, b_j^u\}$ have distance at most s to each other by replacing a_i^v with c_i^v in the paths P and Q.

The fact that vertices a_i^v and a_j^u, and b_i^v and b_j^u, respectively, have distance at most s to each other can be proven similarly as we showed that a_i^v and b_j^u have distance at most s by observing that a_i^v has distance 2 to each vertex a_{i+z}^u with $z \in [-3\ell^*, \lfloor \ell/2 \rfloor + 3\ell^*]$ since a_i^v has neighbors $b_i^u, \ldots, b^u_{i+\lfloor \ell/2 \rfloor}$ and since by Part 2 of Construction 6.20, we have $b_i^u a_{j'}^u \in E(G')$ if and only if $j' = i' + z'$ for some $z' \in [-3\ell^*, 3\ell^*]$.

Furthermore, if ℓ is odd, observe that the above argumentation can also be used to show that the vertex pairs $\{c_i^v, a_j^u\}$ and $\{c_i^v, c_j^u\}$ have distance at most s to each other by replacing a_i^v with c_i^v and replacing a_j^u with c_j^u, respectively, in the paths P and Q.

Hence, S is indeed an s-club. ■

Second, we show that each triangle is contained in sufficiently many triangles.

Claim 8. Each edge in $E(\widetilde{G})$ is contained in at least ℓ triangles which are contained in $\widetilde{G}$.

Proof of Claim. Consider the edge $a_i^v a_{i+j}^v$ for some $j \in [-3\ell^*, 3\ell^*] \setminus \{0\}$. Without loss of generality, assume that $j > 0$. By Part 2 of Construction 6.20, both vertices are adjacent to each vertex $a_{i+i'}^v$ with $i' \in [3\ell^*] \setminus \{j\}$. Hence, both vertices are in at least $3\ell^* - 1 \geq \ell$ triangles.

Furthermore, the statement can be shown analogously for the edges $b_i^v b_{i+j}^v$ (and $c_i^v c_{i+j}^v$ if ℓ is odd) for some $j \in [-3\ell^*, 3\ell^*] \setminus \{0\}$.

Also, the statement can be shown analogously for the edge $a_i^v b_{i+j}^v$ for some $j \in [-3\ell^*, 3\ell^*]$. If $j \geq 0$, then a_{i+z}^v for each $z \in [3\ell^*]$ is a common neighbor of both vertices and thus the edge $a_i^v b_{i+j}^v$ is contained in at least ℓ triangles. The case $j < 0$ can be shown analogously.

For odd ℓ, the statement can be shown analogously for the edges $a_i^v c_{i+j}^v$ and $b_i^v c_{i+j}^v$ for some $j \in [-3\ell^*, 3\ell^*]$, and for the edges $c_i^v c_{i+j}^v$ for some $j \in [-3\ell^*, 3\ell^*] \setminus \{0\}$.

The fact that the edges $a_i^v b_{i+j}^u$ and $c_i^v b_{i+j}^u$ are contained in exactly ℓ triangles follows from the proof of Lemma 6.21.

We conclude that each edge in $E(\widetilde{G})$ is contained in at least ℓ triangles in $\widetilde{G}$. ■

Thus, S is indeed an edge-ℓ-triangle s-club of size k'.

Conversely, let S be an edge-ℓ-triangle s-club of size at least k' in G'. More precisely, let $\widetilde{G}$ be a maximal spanning subgraph of $G'[S]$ which has diameter at most s and such that each edge in $E(\widetilde{G})$ is contained in at least ℓ triangles is $\widetilde{G}$. We show that G contains a clique of size at least k.

First, we show that for each vertex $x \in A_v \cup B_v \cup C_v$ there exists a vertex $y \in A_u \cup B_u \cup C_u$ such that $\operatorname{dist}(x, y) \geq s + 1$ if $uv \notin E(G)$. For this, recall that by construction each two vertices with sub-indices i' and j' are not adjacent if their difference (modulo x) is larger than $3\ell^*$. Also recall that the vertices C_v only exist if ℓ is odd.

Claim 9. In G' we have $\operatorname{dist}(x_i, y_j) \geq s+1$ for each $i \in [0, x]$, $j \coloneqq i + \lfloor \ell/2 \rfloor + 3\ell^*(s-1)$, $x_i \in \{a_i^v, b_i^v, c_i^v\}$, and $y_j \in \{a_j^u, b_j^u, c_j^u\}$ if $uv \notin E(G)$.

Proof of Claim. There are two possible paths from x_i to $y_{i+\lfloor \ell/2 \rfloor + 3\ell^*(s-1)}$ with respect to the indices. First, there is a subsequence of the indices which is increasing $(i, i+1, \ldots, i + \lfloor \ell/2 \rfloor + 3\ell^*(s-1))$. This path has length $\lfloor \ell/2 \rfloor + 3\ell^*(s-1)$. Second, there is a subsequence of the indices which is decreasing $(i, i-1, \ldots, i-i'$, where $-i' \coloneqq \lfloor \ell/2 \rfloor + 3\ell^*(s-1))$. This path has length $3\ell^*(s-1)+1$.

Hence, each path from x_i to $y_{i+\lfloor \ell/2 \rfloor + 3\ell^*(s-1)}$ has to overcome at least $3\ell^*(s-1)+1$ indices. Observe that whenever an edge between A_p (or C_p) and B_q for $p, q \in V(G)$ with $pq \in E(G)$ is traversed, by construction the index can increase/decrease by at most $\lfloor \ell/2 \rfloor$. Now, we use the fact that $uv \notin E(G)$: There are no edges between the vertex gadgets T^v and T^u. Thus, at least two times such a traversal of at most $\lfloor \ell/2 \rfloor$ indices has to be done. Hence, the index i can increase or decrease by at most $2 \cdot \lfloor \ell/2 \rfloor + 3\ell^*(s-2) < 3\ell^*(s-1) + 1$ if at least 2 edge traversals between different vertex gadgets are necessary. Thus, the two vertices x_i and y_j have distance at least $s + 1$. ■

The following statement directly follows from Claim 9.

Claim 10. If $A_v \subseteq S$ (or $C_v \subseteq S$ if ℓ is odd) and $B_u \subseteq S$ then $uv \in E(G)$.

We now use Claims 9 and 10 to show that G contains a clique of size at least k. We distinguish between the cases whether S contains only parts of one of the gadgets A_v, B_v, or C_v or whether S contains all vertices of the gadgets A_v (or B_v, or C_v) completely.

First, assume that for some vertex $v \in V(G)$ we have $A_v \cap S \neq \emptyset$ and $A_v \not\subseteq S$. In the following, we show that S only contains vertices of gadget T^v and from gadgets T^u such that $uv \in E(G)$. Since $A_v \not\subseteq S$, we conclude that in $\widetilde{G}$ we have $N_{\widetilde{G}}(A_v \cap S) \subseteq (B_v \cup C_v)$: Otherwise, since vertex $a_v \in A_v \cap S$ has a neighbor $b_u \in B_u$ and by

Lemma 6.21 we would obtain $A_v \subseteq S$, a contradiction to the assumption $A_v \not\subseteq S$. If $B_v \not\subseteq S$, then by Lemma 6.21 no vertex in B_v can have a neighbor a_i^w or c_i^w for some $w \neq v$. Note that B_v may have neighbors in C_v. Hence, $S \cap T^v$ would be a connected component of size at most $3(x+1)$, a contradiction to the size of S since $k \geq 3$. Thus, we may assume that $B_v \subseteq S$.

Observe that if $a_i^w \in S$ or $c_i^w \in S$ for some $w \in V(G)$ such that $vw \in E(\widetilde{G})$, that is, also $vw \in E(G')$ and thus also $yw \in E(G)$, then we have $A_w \subseteq S$ and $C_w \subseteq S$ by Lemma 6.21 since each vertex a_i^w and c_i^w has a neighbor in B_v. Let $W \coloneqq \{w_1, \ldots, w_t\}$ denote the set of vertices w_j such that $vw_j \in E(G)$ and $A_{w_j}, C_{w_j} \subseteq S$. If $w_x w_y \notin E(G)$ for some $x, y \in [t]$ with $x \neq y$, then $a_0^{w_x}$ and $a_{\lfloor \ell/2 \rfloor + 3\ell^*(s-1)}^{w_y}$ have distance at least $s+1$ by Claim 9. Thus $w_x w_y \in E(G)$ for each $x, y \in [t]$ with $x \neq y$.

Assume towards a contradiction that $a_i^p \in S$ or $c_i^p \in S$ for some $p \in V(G) \setminus W$ with $p \neq v$. Note that $pv \notin E(G)$ since otherwise $p \in W$ by the definition of W. Observe that since $B_v \subseteq S$ we also have $b_{i+\lfloor \ell/2 \rfloor + 3\ell^*(s-1)}^v \in S$. But since $pv \notin E(G)$ we obtain from Claim 9 that $\operatorname{dist}(z_i, b_{i+\lfloor \ell/2 \rfloor + 3\ell^*(s-1)}^v) \geq s+1$ for $z_i = a_i^p$ or $z_i^p = c_i$, a contradiction. We conclude that S does not contain any vertex a_i^p or c_i^p with $p \neq v$ or $p \neq w_j$ for $j \in [t]$.

Next, assume towards a contradiction that $b_i^p \in S$ for some $p \in V(G) \setminus W$ with $p \neq v$. If $pv \notin E(G)$, then b_i^p and $b_{i+\lfloor \ell/2 \rfloor + 3\ell^*(s-1)}^v$ have distance at least $s+1$ again by Claim 9. Thus, we can assume that $pv \in E(G)$. Recall that $b_{i+\lfloor \ell/2 \rfloor + 3\ell^*(s-1)}^v \in S$. As defined by Claim 9, each shortest path from b_i^p to $b_{i+\lfloor \ell/2 \rfloor + 3\ell^*(s-1)}^v$ can swap at most once between different vertex gadgets. In this case, there is exactly one swap from T^p to T^v. From the above we know that $(A_p \cup C_p) \cap S = \emptyset$. Thus, each shortest path from b_i^p to $b_{i+\lfloor \ell/2 \rfloor + 3\ell^*(s-1)}^v$ uses at least one vertex in $A_v \cup C_v$. Since at least one edge with an endpoint in $A_v \cup C_v$ is contained in $E(\widetilde{G})$, we conclude from Lemma 6.21 that $A_v \cup C_v \subseteq S$, a contradiction to the assumption $A_v \not\subseteq S$.

Hence, there is no vertex $p \neq v$ and $p \notin W$ such that $T^p \cap S \neq \emptyset$. In other words, S contains only vertices from the gadget T^v and from gadgets T^u with $vu \in E(G)$. Thus, $S \subseteq T^v \cup \bigcup_{j=1}^t T^{w_j}$. By definition of k', we have $t \geq k-1$ and we conclude that G contains a clique of size at least k. The case that we have $B_v \cap S \neq \emptyset$ and $B_v \not\subseteq S$ or $C_v \cap S \neq \emptyset$ and $C_v \not\subseteq S$ for some vertex $v \in V(G)$ can be handled similarly.

Second, consider the case that for each set A_v with $A_v \cap S \neq \emptyset$ we have $A_v \subseteq S$, that for each set B_v with $B_v \cap S \neq \emptyset$ we have $B_v \subseteq S$, and if ℓ is odd, that for each set C_v with $C_v \cap S \neq \emptyset$ we have $C_v \subseteq S$. Let $W_A \coloneqq \{w_A^j \in V(G) \mid A_{w_j} \subseteq S\}$, $W_B \coloneqq \{w_B^j \in V(G) \mid B_{w_j} \subseteq S\}$, and $W_C \coloneqq \{w_C^j \in V(G) \mid C_{w_j} \subseteq S\}$. If $W_A = \emptyset$ or $W_B = \emptyset$ or $W_C = \emptyset$ (recall that these vertices only exist if ℓ is odd),

then each connected component in $G'[S]$ has size at most $2(x+1) < k'$. Thus, we may assume that $W_A \neq \emptyset$, $W_B \neq \emptyset$, and $W_C \neq \emptyset$. By Claim 10, we have $w_A^i w_B^j \in E(G)$ for each $w_A^i \in W_A$ and $w_B^j \in W_B$ and, if ℓ is odd, also $w_C^i w_B^j \in E(G)$ for each $w_C^i \in W_C$ and $w_B^j \in W_B$. Furthermore, by Claim 9, we have $w_B^j w_B^{j'} \in E(G)$ for distinct $w_B^j, w_B^{j'} \in W_B$, $w_A^i w_A^{i'} \in E(G)$ for distinct $w_A^i, w_A^{i'} \in W_A$, and if ℓ is odd we also have $w_C^i w_C^{i'} \in E(G)$ for distinct $w_C^i, w_C^{i'} \in W_C$ and $w_A^i w_C^{i'} \in E(G)$ for $w_A^i \in W_A$ and $w_C^{i'} \in W_C$.

Recall that $k' = yk$ where $y \coloneqq 2(x+1)k$ if ℓ is even and $y \coloneqq (2(x+1)+6s-5)k$ if ℓ is odd and that for each vertex $v \in V(G)$ we have $|A_v| + |B_v| + |C_v| = y$. Hence, we obtain that $\max(|W_A|, |W_B|, |W_C|) \geq k$. Without loss of generality, we assume that $|W_A| \geq k$. Then since $w_A^i w_A^{i'} \in E(G)$ for distinct $w_A^i, w_A^{i'} \in W_A$ we conclude that G contains a clique of size at least k. □

6.3 Seeded s-Club

In this section we study the parameterized complexity of Seeded s-Club with respect to the standard parameter solution size k. Recall that in this problem we aim to find an s-club containing a given seed of vertices. Here, we assume that $|W| < k$ since otherwise the problem can be solved in polynomial time.

6.3.1 Tractable Cases

For clique seeds, we provide the following kernel. Note that here we present a kernel and not only a Turing kernel.

Theorem 6.22. Seeded s-Club *admits a kernel with $\mathcal{O}(k^{2|W|+1})$ vertices if W induces a clique.*

Note that this kernel has polynomial size if W has constant size. In the following, assume that $G[W]$ is a clique. To prove the kernel, we first remove all vertices with distance at least $s+1$ to any vertex in W. Second, we show that if the remaining graph, that is $N_s[W]$, is sufficiently large, then (G, W, k) is a trivial yes-instance.

Reduction Rule 6.3. *Let (G, W, k) be an instance of* Seeded s-Club. *If G contains a vertex u such that dist$(u, w) \geq s+1$ for some $w \in W$, then remove u.*

Clearly, Reduction Rule 6.3 is correct and can be applied in polynomial time. Next, we show that if the remaining graph is sufficiently large, then (G, W, k) is a yes-instance of Seeded s-Club.

Lemma 6.23. *An instance* (G, W, k) *of* Seeded s-Club *with* $|N_{s-1}[W]| \geq k^2$ *is a yes-instance.*

To prove Lemma 6.23 for $s \geq 3$ we need the following technical lemmas.

Lemma 6.24. *An instance* (G, W, k) *of* Seeded s-Club *with* $s \geq 3$ *is a yes-instance if* $|N_{\lfloor (s+1)/2 \rfloor - 1}[W]| \geq k$.

Proof. By definition $W \subseteq N_{\lfloor (s+1)/2 \rfloor - 1}[W]$ and $|N_{\lfloor (s+1)/2 \rfloor - 1}[W]| \geq k$. Thus, it remains to show that $N_{\lfloor (s+1)/2 \rfloor - 1}[W]$ is an s-club. For this, consider a pair of vertices $u, v \in N_{\lfloor (s+1)/2 \rfloor - 1}[W]$. Observe that by definition $\mathrm{dist}(u, W) \leq \lfloor (s+1)/2 \rfloor - 1$ and $\mathrm{dist}(v, W) \leq \lfloor (s+1)/2 \rfloor - 1$. Since W is a clique, we have $\mathrm{dist}(u, v) \leq (\lfloor (s+1)/2 \rfloor - 1) + 1 + (\lfloor (s+1)/2 \rfloor - 1) \leq s$. Hence, the lemma follows. □

Note that the assumption $s \geq 3$ in Lemma 6.24 is necessary to guarantee that $\lfloor (s+1)/2 \rfloor - 1 \geq 1$. Next, we show that if a vertex in $N_{\lfloor (s+1)/2 \rfloor - 1}(W)$ has many vertices close to it, then (G, W, k) is a yes-instance.

Lemma 6.25. *An instance* (G, W, k) *of* Seeded s-Club *with* $s \geq 3$ *and with* $|N_{\lfloor s/2 \rfloor}[v]| \geq k$ *for some vertex* $v \in N_{\lfloor (s+1)/2 \rfloor - 1}(W)$ *is a yes-instance.*

Proof. Let v be a vertex as specified in the lemma. By definition of v, there exists a path $P \coloneqq (q_0, q_1, \ldots, q_{\lfloor (s+1)/2 \rfloor - 1})$ of length $\lfloor (s+1)/2 \rfloor - 1$ in G such that $q_{\lfloor (s+1)/2 \rfloor - 1} = v$, $q_0 \in W$, and $q_i \in N_i(q_0)$. We show that $S \coloneqq N_{\lfloor s/2 \rfloor}[v] \cup W \cup P$ is an s-club of size k containing W. Clearly, $W \subseteq S$ and $|S| \geq k$. Thus, it remains to show that S is an s-club.

Consider a vertex $w \in W$. Vertex w has distance at most $i + 1$ to vertex q_i. In particular, $\mathrm{dist}(w, v) \leq \lfloor (s+1)/2 \rfloor$. Since each vertex $u \in N_{\lfloor s/2 \rfloor}[v]$ has distance at most $\lfloor s/2 \rfloor$ to v we obtain that $\mathrm{dist}(w, u) \leq \lfloor (s+1)/2 \rfloor + \lfloor s/2 \rfloor = s$. By similar arguments we can also show that vertex q_i for $i \in [\lfloor (s+1)/2 \rfloor - 1]$ has distance at most s to each vertex in S.

Finally, consider two vertices $x, y \in N_{\lfloor s/2 \rfloor}[v]$. Note that $\mathrm{dist}(x, v) \leq \lfloor s/2 \rfloor$ and also $\mathrm{dist}(y, v) \leq \lfloor s/2 \rfloor$ and thus $\mathrm{dist}(x, y) \leq s$.

Thus, S is indeed an s-club. □

With those two lemmas we are now able to prove Lemma 6.23.

Proof of Lemma 6.23. First, we consider the case $s = 2$. Since $|W| \leq k$ and since $N[W] \geq k^2$, it is sufficient to show that (G, W, k) is a yes-instance if $|N[w]| \geq k$ for some $w \in W$. Since all vertices in $N(w)$ have the common neighbor w, we conclude that $N[w]$ is a 2-club. Also, since W is a clique, we have $W \subseteq N[w]$. The size bound of $N_{s-1}[W]$ follows from $|N[w]| \geq k$. Thus, (G, W, k) is a yes-instance.

Second, we consider the case $s \geq 3$. Observe that

$$N_{s-1}[W] = N_{\lfloor (s+1)/2 \rfloor - 1}[W] \cup \bigcup_{v \in N_{\lfloor (s+1)/2 \rfloor - 1}[W]} N_{\lfloor s/2 \rfloor}[v].$$

By Lemma 6.24, (G, W, k) is a yes-instance if $|N_{\lfloor (s+1)/2 \rfloor - 1}[W]| \geq k$. Furthermore, by Lemma 6.25, (G, W, k) is a yes-instance if $|N_{\lfloor s/2 \rfloor}[v]| \geq k$ for some $v \in N_{\lfloor (s+1)/2 \rfloor - 1}(W)$. Thus, by the above equality we conclude that (G, W, k) is a yes-instance. □

Finally, we bound the size of $N_s(W)$. There we assume that $|N_{s-1}[W]| < k^2$ by Lemma 6.23 and that Reduction Rule 6.3 is applied.

Lemma 6.26. *An instance (G, W, k) of* Seeded s-Club *such that $|N_s(W)| \geq k^{2|W|+1}$ which is reduced with respect to Reduction Rule 6.3 is a yes-instance.*

Proof. Since Reduction Rule 6.3 has been applied exhaustively, each vertex $p \in N_s(W)$ has distance exactly s to each vertex in W. In other words, for each vertex $w_\ell \in W$ there exists a vertex $u^\ell_{s-1} \in N_{s-1}(w_\ell)$ such that $pu^\ell_{s-1} \in E(G)$. Note that $N_{s-1}(w_\ell) \subseteq N_{s-1}[W]$. Moreover, by Lemma 6.23 we may assume that $|N_{s-1}[W]| < k^2$. In particular: $|N_{s-1}(W)| < k^2$. Since $|N_s(W)| \geq k^{2|W|+1}$, by the pigeonhole principle there exists a set $\{u^1_{s-1}, u^2_{s-1}, \ldots, u^{|W|}_{s-1}\}$ with $u^\ell_{s-1} \in N_{s-1}(w_\ell)$ for $\ell \in [|W|]$ such that the set $P \coloneqq N_s(W) \cap \bigcap_{\ell \in [|W|]} N(u^\ell_{s-1})$ has size at least k. The size bound follows from the observation that each $N_{s-1}(w_\ell)$ has size at most k^2 and we have exactly $|W|$ many of these sets. By the definition of vertex u^ℓ_{s-1}, there exists for each $i \in [s-2]$ a vertex $u^\ell_i \in N_i(w_\ell)$ such that $w_\ell, u^\ell_1, \ldots, u^\ell_{s-1}$ is a path of length $s-1$ in G. We define the set $U \coloneqq \{u^\ell_i \mid \ell \in [|W|], i \in [s-1]\}$. Next, we show that $Z \coloneqq P \cup W \cup U$ induces an s-club.

First, observe that all vertices in P have distance at most 2 to each other since they have the common neighbor u^1_{s-1}. Second, note that the vertices w_ℓ, $u^\ell_1, \ldots, u^\ell_{s-1}$, $p, u^j_{s-1}, \ldots, u^j_1, w_j$ form a cycle with $2s+1$ vertices, for each $p \in P$ and each two indices $j, \ell \in [|W|]$. Each vertex in this cycle has distance at most s to each other vertex in that cycle. Hence, Z is indeed an s-club. □

Recall that Lemma 6.23 showed that the number of vertices with distance *at most* $s-1$ to W is bounded by k^2. Together with Lemma 6.26 now Theorem 6.22 is proven.

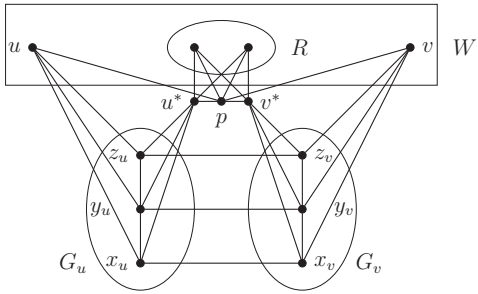

Figure 6.6: Illustration of Construction 6.28.

6.3.2 Intractable Cases

Now, we show hardness for some of the remaining cases.

Theorem 6.27. *Let H be a fixed graph.* Seeded s-Club *is W[1]-hard parameterized by k even if $G[W]$ is isomorphic to H, when*

- *$s = 2$ and H contains at least two non-adjacent vertices, or if*
- *$s \geq 3$ and H contains at least two connected components.*

Hardness for $s = 2$. First, we prove hardness for $s = 2$ when H contains at least one non-edge. For an illustration of Construction 6.28 we refer to Figure 6.6.

Construction 6.28. Let (G, k) be an instance of Clique. We construct an equivalent instance (G', k') of Seeded s-Club as follows. Initially, we add the set W to G', and add edges such that $G'[W]$ is isomorphic to H. Since H is not a clique, there exist two vertices $u, v \in V(H)$ such that $uv \notin E(H)$. Let $R \coloneqq W \setminus \{u, v\}$. Next, we add two copies G_u and G_v of G to G', make u adjacent to each vertex in $V(G_u)$, and make v adjacent to each vertex in $V(G_v)$. For $x \in V(G)$ we denote with x_u and x_v the copies of x in G_u and G_v, respectively. Next, we add the edge $x_u x_v$ for each $x \in V(G)$. Furthermore, we add a new vertex p and make it adjacent to each vertex in W. Next, we add a new vertex u^* adjacent to p, each vertex in $V(G_u)$, and each vertex in R. Analogously, we add a new vertex v^* which is adjacent to p, each vertex in $V(G_v)$, and each vertex in R. Finally, we set $k' \coloneqq 2k + |W| + 3$.

Now, we prove the correctness of Construction 6.28.

Lemma 6.29. *For any graph H which is not a clique,* Seeded 2-Club *parameterized by k is W[1]-hard if the subgraph induced by W is isomorphic to H.*

Proof. We prove that G contains a clique of size k if and only if G' contains a 2-club S containing W of size $k' = 2k + |W| + 3$.

Let K be a clique of size k in G and let K_u and K_v be the copies of K in G_u and G_v. We argue that $S \coloneqq K_u \cup K_v \cup W \cup \{u^*, v^*, p\}$ is a 2-club of size at least k' containing W. Clearly, $|S| = k'$ and S contains W. Thus, it remains to show that S is a 2-club in G'.

First, we show that each vertex in R has distance at most 2 to each vertex in S: All vertices in R have the common neighbors p, u^*, and v^*. Since u and v are neighbors of p, each vertex in $V(G_u)$ is a neighbor of u^*, and since each vertex in $V(G_v)$ is a neighbor of v^*, we conclude that each vertex in R has distance at most 2 to any vertex in S.

Second, we show that each vertex in K_u has distance at most 2 to each vertex in $S \setminus R$. Observe that $\{u, u^*, x_v\} \cup K_u \subseteq N[x_u]$ for each vertex $x_u \in K_u$. Hence, x_u has distance at most 2 to p via u^*,u, v and to v^* via x_v, and to each vertex in K_v via the corresponding vertex in K_u. By symmetric arguments the statement also holds for each vertex in K_v.

Finally, each pair of vertices of $\{p, u, u^*, v, v^*\}$ has distance at most 2 to each other since $u, u^*, v, v^* \in N(p)$.

Thus, S is indeed a 2-club.

Conversely, suppose that G' contains a 2-club S of size at least $2k + |W| + 3$ which contains all vertices of W. Observe that we have $N(x_u) \cap N(v) = \{x_v\}$ for each vertex $x_u \in V(G_u)$, and symmetrically $N(x_v) \cap N(u) = \{x_u\}$ for each vertex $x_v \in V(G_v)$. Hence, $x_u \in S$ if and only if $x_v \in S$. Let $K_u \coloneqq S \cap V(G_u)$. By definition of k' we obtain that $|K_u| \geq k$. Assume towards a contradiction that K_u contains a pair of non-adjacent vertices x_u and y_u. By the argumentation above we obtain $y_v \in S$. Now, observe that $N(x_u) = \{u, u^*, x_v\} \cup \bigcup\{z_u \mid xz \in E(G)\}$ and that $N(y_v) = \{v, v^*, y_u\} \cup \bigcup\{z_v \mid yz \in E(G)\}$. Since $xy \notin E(G)$ we thus obtain $N(x_u) \cap N(y_v) = \emptyset$, a contradiction. Thus, G contains a clique of size k. □

Hardness for seeds with at least two connected components and $s \geq 3$. Now, we show W[1]-hardness for the case $s \geq 3$ when the seed contains at least two connected components. Fix a graph H with at least two connected components. We show W[1]-hardness for $s \geq 3$ even if $G[W]$ is isomorphic to H.

Construction 6.30. Let (G, k) be an instance of Clique. We construct an equivalent instance (G', k') of Seeded s-Club as follows. Initially, we add the set W

to G', and add edges such that $G'[W]$ is isomorphic to H. Let D_1 be one connected component of $G'[W]$. By assumption, $D_2 \coloneqq W \setminus D_1$ is not empty. Next, we add two copies G_1 and G_2 of G to G'. Then, we add edges to G' such that each vertex in D_1 is adjacent to each vertex in $V(G_1)$ and such that each vertex in D_2 is adjacent to each vertex in $V(G_2)$. Furthermore, we add a path $(p_1, \ldots, p_{s-1})$ consisting of exactly $s-1$ new vertices to G', make p_1 adjacent to each $u \in D_1$, and make p_{s-1} adjacent to each $v \in D_2$. By $P \coloneqq \{p_i \mid i \in [s-1]\}$ we denote the set of these newly added vertices. Now, for each $x \in V(G)$ we do the following. Consider the copies $x_1 \in V(G_1)$ and $x_2 \in V(G_2)$ of vertex $x \in V(G)$. We add a path $(x_1, q_1^x, \ldots, q_{s-2}^x, x_2)$ consisting of $s-2$ new vertices to G'. By $Q_x \coloneqq \{q_i^x \mid i \in [s-2]\}$ we denote the set of the new internal path vertices. Finally, we set $k' \coloneqq sk + |W| + s - 1$.

Now, we prove the correctness of Construction 6.30.

Lemma 6.31. *Let H be a fixed graph with at least two connected components.* Seeded s-Club *for $s \geq 3$ parameterized by k is W[1]-hard even if $G[W]$ is isomorphic to H.*

Proof. We show that G contains a clique of size k if and only if G' contains an s-club containing W of size at least $k' = sk + |W| + s - 1$.

Let K be a clique of size k in G. Furthermore, let K_1 and K_2 denote the copies of K in G_1 and G_2, respectively. We show that $S \coloneqq W \cup P \cup K_1 \cup K_2 \cup \bigcup_{x \in K} Q_x$ is an s-club containing W of size at least k'. Clearly, $|S| = k'$ and S contains W. Thus, it remains to verify that S is an s-club in G'. Note that since each vertex in $V(G_1)$ is adjacent to each vertex in D_1, each two vertices in D_1 have distance at most 2, and similarly each two vertices in $V(G_1)$ have distance at most 2. Analogously, we can show that each two vertices in D_2 and each two vertices in $V(G_2)$ have distance at most 2. Furthermore, the vertices $(x, p_1, p_2, \ldots, p_{s-1}, y, u_2, q_{s-2}^u, \ldots, q_1^u, u_1)$ for each vertex $x \in D_1$, each vertex $y \in D_2$, and each vertex $u \in K$ form a C_{2s+1}, a cycle with $2s+1$ vertices. Observe that also the vertices $(u_1, q_1^u, \ldots, q_{s-2}^u, u_2, v_2, q_{s-2}^v, \ldots, q_1^v, v_1)$ form a C_{2s} for each two vertices $u, v \in K$. Since all remaining distances are covered by these two kinds of cycles, we conclude that S is indeed an s-club.

Conversely, suppose that G' contains an s-club S containing W of size at least k'. Let $Q'_v \coloneqq \{v_1, q_1^v, \ldots, q_{s-2}^v, v_2\}$ for each $v \in V(G)$. We show that $Q'_v \cap S \neq \emptyset$ if and only if $Q'_v \subseteq S$. Assume towards a contradiction, that $Q'_v \cap S \neq \emptyset$ for some $v \in V(G)$ such that $Q'_v \not\subseteq S$. If $v_1 \notin S$, and also $v_2 \notin S$, then no vertex in $S \cap Q'_v$ is connected to any vertex in $S \setminus Q'_v$. Hence, we can assume without loss of generality that $v_1 \in S$. Note that $N(D_2) = V(G_2) \cup \{p_{s-1}\}$. Furthermore, observe that $\operatorname{dist}(v_1, p_{s-1}) = s$, that $\operatorname{dist}(v_1, q_{s-2}^v) = s-2$, and that $\operatorname{dist}(v_1, q_{s-2}^u) \geq s-1$ for each $u \in V(G) \setminus \{v\}$.

Thus, the unique path of length at most s from v_1 to D_2 contains all vertices in Q'_v. Hence, $Q'_v \cap S \neq \emptyset$ if and only if $Q'_v \subseteq S$. By the definition of k' we may thus conclude that $Q'_v \subseteq S$ for at least k vertices $v \in V(G)$.

Now, assume towards a contradiction that $Q'_u \subseteq S$ and $Q'_v \subseteq S$ such that $uv \notin E(G)$. We consider the vertices v_1 and u_2. Observe that by construction each path from v_1 to u_2 containing any vertex p_i has length at least $s+2$. Hence, each shortest path from v_1 to u_2 contains the vertex set of Q'_w for some $w \in V(G)$. Since the path induced by each Q'_w has length $s-1$, we conclude that $w = u$ or $w = v$. Assume without loss of generality that $w = v$. Hence, the $(s-1)$th vertex on the shortest path from u_2 to v_1 is v_2. Since $uv \notin E(G)$ we have by construction that $u_2v_2 \notin E(G')$. Hence, $\text{dist}(v_1, u_2) \geq s+1$, a contradiction to the fact that S is an s-club. Thus, $\{v \mid Q'_v \subseteq S\}$ is a clique of size at least k in G. □

6.4 Conclusion

We provided a complexity dichotomy for Vertex Triangle s-Club and Edge Triangle s-Club for the standard parameter solution size k with respect to s and ℓ. Furthermore, we also provided a complexity dichotomy for Seeded 2-Club for k in terms of the structure of $G[W]$. For Seeded s-Club with $s \geq 3$ we provided an FPT-algorithm with respect to k when $G[W]$ is a clique and we showed W[1]-hardness for k when $G[W]$ contains at least 2 connected components. Hence, an immediate open question is the parameterized complexity of Seeded s-Club for $s \geq 3$ when $G[W]$ is connected but not a clique. One aim should be to also provide a dichotomy for k for Seeded s-Club with $s \geq 3$ for all possible structures of $G[W]$. It is particularly interesting to study seeds of constant size since this seems to be the most interesting case for applications.

Furthermore, for future work it seems interesting to study the complexity of the considered variants of s-Club with respect to further parameters, as it was done for 2-Club [110]. For example it is interesting to study these problems with respect to structural parameters of the input graph G such as the treewidth of G or the feedback edge set number. 2-Club admits an FPT-algorithm with respect to the treewidth [109]. Note that even the arguments for the larger parameter vertex cover [109] cannot applied directly for Vertex Triangle s-Club and Edge Triangle s-Club: First all possibilities of the intersection of the vertex cover with the solution was brute forced and second all vertices with a too large distance to the current sub-solution were deleted. This algorithm fails for these two problems since this algorithm some vertices may not be in sufficiently many triangles. Furthermore, 2-Club admits a polynomial kernel with respect to the feedback edge set

number [109]. Also, the arguments of the linear kernel of Hartung et al. [109] can not be used directly for the three problems studied in this chapter since one argument to obtain this kernel is that $\Delta < k$ which is not true for the three problems studied in this chapter.

It is also interesting to study other problems for detecting communities with seed constraints. The first problem which comes in mind could be Clique with seed constraint. While this problem could be interesting in applications, it is not interesting from an algorithmic point of view: If a vertex set W is seeded, we have to find the largest clique in the common neighborhood of W. For this task the standard clique algorithms can be used. Hence, from an algorithmic point of view it is more interesting to study other clique relaxations with seed constraints, for example s-Plex or s-Defective Clique. These problems are also NP-hard, since an algorithm for the case when $|W| = 1$ can be used as a black box to solve the unseeded variants. It is interesting to investigate the parameterized complexity of these problems.

One example of such a seeded clique relaxation is the combination of the studied problems in this chapter. More precisely, one could study the following problem.

Seeded Vertex Triangle s-Club
Input: An undirected graph $G = (V, E)$, a subset $W \subseteq V$, and two integers $k, \ell \geq 1$.
Question: Does G contain an s-club S of size at least k such that S fulfills the vertex-ℓ-triangle property and $W \subseteq S$?

Also the problem Seeded Edge Triangle s-Club is interesting. In this problem, we put the triangle constraint on the edges instead of the vertices. The NP-hardness of Seeded Vertex Triangle s-Club and Seeded Edge Triangle s-Club follows directly from the fact that an algorithm for the case where $|W| = 1$ can be used as a black box to solve Vertex Triangle s-Club. Furthermore, our W[1]-hardness results for the parameter k for Vertex Triangle s-Club and Edge Triangle s-Club from Theorems 6.7 and 6.19 can be extended to provide W[1]-hardness also for Seeded Vertex Triangle s-Club and Seeded Edge Triangle s-Club: We add a new vertex gadget which is seeded to the constructed graph and update the budget accordingly. Thus, for the standard parameter k only the cases for which Vertex Triangle s-Club and Edge Triangle s-Club are FPT are interesting, that is, $\ell = 1$ and $s \geq 4$ for the vertex variant, and $\ell = 1$ and $s \geq 2$ for the edge variant. But the FPT-algorithms for these cases cannot be adapted that easily: These FPT-algorithms relied on Turing kernels in which we

showed that if some vertex has a large neighborhood, then the instance is a yes-instance. Such an argument is not possible anymore because we now also have to consider the distance to the seed vertices.

Additionally, it is interesting to study the parameterized complexity of further robust variants of s-Club such as t-Hereditary s-Club [145, 193], t-robust s-clubs [222], and t-connected s-clubs [231, 145]. It was shown these three problems admit an FPT-algorithm with respect to the maximum degree Δ [145]. Furthermore, these tree problems also admit an FPT-algorithm with respect to $n - k$, the number of deleted vertices [145]. It remains an open question to study the parameterized complexity of these three problems with respect to k, the solution size. It is very likely that these three problems are W[1]-hard with respect to k since our results for Vertex Triangle s-Club, Edge Triangle s-Club and Seeded s-Club showed that the FPT results for s-Club with respect to k are quite brittle. For following work it is also interesting to study the seeded versions of these problems.

Chapter 7

Experiments for Triangle 2-Clubs

In this chapter we engineer an exact solver for the two problems Vertex Triangle 2-Club and Edge Triangle 2-Club studied in Chapter 6 in terms of their parameterized complexity with respect to the standard parameter solution size k. Recall that a vertex set $S \subseteq V$ is a *vertex-ℓ-triangle 2-club* in G if $G[S]$ has diameter at most two and every vertex of S is in at least ℓ triangles in $G[S]$ [3] (see also Definition 6.2). This model leads to the following optimization problem.

Vertex Triangle 2-Club
Input: An undirected graph $G = (V, E)$ and an integer ℓ.
Task: Find a maximum-cardinality vertex-ℓ-triangle 2-club.

Furthermore, recall that a set S is an *edge-ℓ-triangle 2-club* when $G[S]$ has a spanning subgraph $\widehat{G} = (S, \widehat{E})$ such that $\widehat{G}$ has diameter at most 2 and every edge of $\widehat{E}$ is in at least ℓ triangles in $\widehat{G}$ (see also Definition 6.3). This model leads to the following optimization problem.

Edge Triangle 2-Club
Input: An undirected graph $G = (V, E)$ and an integer ℓ.
Task: Find a maximum-cardinality edge-ℓ-triangle 2-club.

Both problems were introduced to overcome a major downside of the 2-club model: Often a vertex of maximum degree together with all its neighbors is also the largest 2-club in the graph. As a consequence, maximum 2-clubs fare poorly when it comes to other typical properties of communities such as having high density or being robust against vertex or edge failures. As discussed in Chapter 6 both the vertex-ℓ-triangle 2-club and edge-ℓ-triangle 2-club have many desirable properties: For example, both models imply a clustering coefficient larger than zero. Furthermore, an edge-ℓ-triangle 2-club is still connected after ℓ edge deletions (Proposition 6.4).

In the past 20 years, many exact solvers for s-Club were developed. For example, Bourjolly et al. [31] not only provided NP-hardness for finding a largest s-club, they also presented an ILP formulation and an exact branch-and-bound algorithm. This branch-and-bound algorithm was later improved by Chang et al. [43] by providing new reduction rules and upper bounds within the search tree.

A particularly important case is $s = 2$, where we search for a vertex set that induces a subgraph of diameter at most 2. Also, several ILPs were developed which can solve 2-Club efficiently [14, 185, 203]. Furthermore, it was demonstrated that branch-and-bound algorithms are competitive with ILPs to solve 2-Club [43, 109, 145]. As noted above, these studies showed an undesirable behavior of maximum 2-clubs: in most instances, the largest 2-club consists of a vertex of maximum degree and its neighbors.

In this chapter, we study whether exact solutions for both problems can be efficiently found in practice. More precisely, we provide branch-and-bound algorithms for the standard parameter solution size k based on the standard Turing kernels for k for 2-Club. For Vertex Triangle 2-Club and $\ell = 1$, a first ILP to find optimal solutions was presented by Carvalho and Almeida [41]. This ILP formulation was then generalized by Almeida and Brás [3] to arbitrary values of ℓ. They showed that their ILP can solve medium-size instances (up to 10 000 vertices) for all $\ell \leq 6$ efficiently (in less than 10 minutes) [3]. Both ILP-based algorithms, which more generally solve Vertex Triangle s-Club for arbitrary values of s, consist of three main features: First, an efficient separation that computes valid inequalities in case a vertex set violates the vertex-ℓ-triangle property or the diameter property is violated. Second, a data reduction rule that removes vertices which are not in sufficiently many triangles is described. Finally, an efficiently computable lower bound, called the *Neighborhood Lower Bound* in this work, is described. The Neighborhood Lower Bound is an adaption of the classic star heuristic for 2-club [3]. The star heuristic finds the vertex with maximum degree as a valid 2-club solution which very often turns out to be the optimal solution. In the context of Vertex Triangle 2-Club, the Neighborhood Lower Bound aims to find the largest vertex-ℓ-triangle 2-club contained in the neighborhood of some vertex $v \in V(G)$. The ILP formulations are so far the only computational study on the two problems.

Our Results. For brevity, we use the term triangle 2-clubs to simultaneously refer to the vertex-ℓ-triangle 2-club and the edge-ℓ-triangle 2-club properties for all ℓ.

We show that, using a combinatorial branch-and-bound algorithm combined with problem-specific data reduction rules and lower bounds, we are able to solve large sparse instances of Vertex Triangle 2-Club with up to 290 000 vertices

and 990 000 edges. Compared to the previous computational experiments of Almeida and Brás [3], which were run only for $\ell \leq 6$, we show that Vertex Triangle 2-Club can be solved efficiently for ℓ up to 100. Our implementation outperforms the ILP of Almeida and Brás [3] on all except one instance considered previously. In addition, our implementation also solves Edge Triangle 2-Club. Furthermore, our algorithm is faster for Edge Triangle 2-Club than for Vertex Triangle 2-Club. In contrast, for Edge Triangle 2-Club, we expect ILP-based formulations to be slower than for Vertex Triangle 2-Club: direct formulations need variables that encode the presence of edges which is not necessary for the vertex-triangle property.

The overall approach of our branch-and-bound algorithm is similar to previous ones for 2-Club [109] and other robust models of 2-Club like t-Hereditary 2-Club [145]: First, perform data reduction and compute a lower bound. Second, consider the sufficiently large 2-neighborhoods of G, that is, the 2-neighborhoods that are larger than the lower bound, one by one. On each 2-neighborhood, perform a branching that identifies a vertex that is in conflict with at least one other vertex, for example since their distance within the 2-neighborhood is too large, that is, at least 3.

Naturally, the details of the branch-and-bound algorithm differ from previous ones when it comes to incorporating the triangle properties. We first use data reduction rules that establish the triangle properties. The direct rules that delete vertices or edges whenever they are not in sufficiently many triangles have been described previously [3, 88]. We add further simple-degree-based rules to speed up the data reduction. We identify further data reduction rules that delete edges and vertices, since including them in a triangle 2-club would lead to a violation of the triangle properties. These rules particularly apply to the setting during branching when some vertices are already marked as being part of the sought triangle 2-club.

We use two heuristics for computing lower bounds. The first one is the known *Neighborhood Lower Bound*. This lower bound performs very well on many instances but not on all instances. In particular for intermediate values of ℓ, the lower bound quality decreases. This prompts us to develop a new heuristic, called *Greedy Lower Bound*, which considers the 2-neighborhoods, as the branching algorithm does. Instead of performing a branching it greedily deletes vertices that are in a conflict. We show that the combination of the two lower bounds substantially outperforms the *Neighborhood Lower Bound* in terms of solution quality. Since the running time of the Greedy Lower Bound is quite high, it leads to substantial running time improvements only for Edge Triangle 2-Club. Besides the experimental evaluation, we also analyze the theoretical worst-case running time of the lower bounds depending

on the degeneracy and the total size of the input graph. Furthermore, we show that if an optimal solution S for Vertex Triangle 2-Club or Edge Triangle 2-Club contains a universal vertex, then the Neighborhood Lower Bound finds S.

Our experiments show that the running time decreases for larger values of ℓ. The main reason for this behavior is our initial data reduction which deletes vertices with too small degree. These results are in sharp contrast to our theoretical results from Chapter 6: We showed that Edge Triangle 2-Club is only FPT for k if $\ell = 1$ and that Vertex Triangle 2-Club is W[1]-hard for k for any $\ell \geq 1$. In other words, these sharp theoretical boundaries cannot be observed in practice. A main reason for this is the quality of our lower bounds and the effectiveness of our initial data reduction rules.

We then study how the parameter ℓ influences central properties of the returned triangle 2-clubs, as done in previous work [3]. In a nutshell, we confirm that increasing ℓ usually leads to high density and high global and local clustering coefficients, also on larger networks, where the general tendency is that maximum (triangle) 2-clubs are less dense. Furthermore, our experiments show that already for $\ell = 1$ the density and the local and global clustering coefficient are relatively high. Since increasing ℓ usually does not influence the running time negatively, this gives a good adjustable parameter for balancing the emphasis on the low-diameter property with other cohesiveness properties.

7.1 The Branching Algorithm

Our algorithms to solve Vertex Triangle 2-Club and Edge Triangle 2-Club are based on the observation that the solution is contained in the 2-neighborhood $N_2[v]$ of some vertex v of the input graph [206]. Consequently, we solve a given instance (G, ℓ) by finding the solutions of all local instances (G_v, ℓ), where G_v is the subgraph induced by $N_2[v]$. Afterwards, we return the solution of maximum size. Our algorithm stores the size k of a largest triangle 2-club detected so far. Then, our algorithm tries to find triangle 2-clubs of size at least $k + 1$.

To solve the local instances, we describe two branching algorithms. Furthermore, we provide several reduction rules that we apply in our implementation.

7.1.1 Basic Reduction Rules and Naive Branching Algorithm

We first describe a naive strategy to solve Vertex Triangle 2-Club and Edge Triangle 2-Club. The algorithm relies on a simple reduction rule and a branching strategy. Let (G, ℓ) be an instance of Vertex Triangle 2-Club and let v be a

vertex of G. If v is contained in less than ℓ triangles of G, it cannot be part of any vertex set S that forms a vertex-ℓ-triangle 2-club. It is easy to see that this vertex can safely be removed from the input graph. Observe that the same holds for Edge Triangle 2-Club, since a set that fulfills the edge-ℓ-triangle property also fulfills the vertex-ℓ-triangle property. For an instance (G, ℓ) of Edge Triangle 2-Club we can also remove some edges: Let e be an edge of G that is contained in less than ℓ triangles. Furthermore, let S be an edge-ℓ-triangle 2-club in G and let $\widehat{G} = (S, \widehat{E})$ be a corresponding subgraph with diameter at most 2, where every edge of $\widehat{E}$ is in at least ℓ triangles in $\widehat{G}$. Then, $e \notin \widehat{E}$ and therefore, the edge e can safely be removed from the input instance.

Both observations described above lead to a reduction rule in which we can safely remove vertices (or edges, respectively) that are not contained in enough triangles.

Reduction Rule 7.1 (Low-Triangle Rule (LTR))**.** *a) In case of* Vertex Triangle 2-Club*, remove all vertices that are in less than ℓ triangles.*

b) In case of Edge Triangle 2-Club*, remove all edges that are in less than ℓ triangles and remove isolated vertices.*

We now bound the running time of exhaustively applying this rule. To obtain a bound that explains why the rule runs relatively fast on sparse instances, we formulate it in terms of the number of edges m and the degeneracy d of the input graph.

Lemma 7.1. *The LTR can be performed exhaustively in $\mathcal{O}(m \cdot d)$ time.*

Proof. Using the triangle enumeration algorithm of Chiba and Nishizeki [46], we can enumerate all triangles in $\mathcal{O}(m \cdot d)$ time. While enumerating the triangles, we can compute for each vertex w and each edge e of G a list of all triangles containing w or e and their number.

After this prepossessing, a vertex or an edge to which the rule applies can be determined in $\mathcal{O}(1)$ time as long as the triangle counters for the vertices or edges are updated after each deletion. To update the triangle counter, we traverse the triangle list for the deleted vertex or edge and decrement the respective counter for all vertices or edges that are contained in the triangle. □

To decrease the effort of counting triangles, we provide another rule that exploits the degree of vertices in the input graph to identify some vertices that are not contained in at least ℓ triangles. Observe that a vertex v is contained in at most $\binom{\deg(v)}{2}$ triangles, which is the case if $N[v]$ is a clique. Therefore, a vertex v with $\binom{\deg(v)}{2} < \ell$

Algorithm 7.1: `NaiveBranching` Algorithm to solve Vertex Triangle 2-Club or Edge Triangle 2-Club, respectively.

Input: An instance $(G = (V, E), \ell)$.
Output: A maximum-cardinality triangle 2-club.
1 Apply Reduction Rules 7.2 and 7.1 on the input instance
2 **if** *all* $v \in V$ *are pairwise compatible,* **then**
3 | return V
4 **else**
5 | Find two vertices u and w that are incompatible
6 | $S_1 \coloneqq$ `NaiveBranching`$(G - u, \ell)$
7 | $S_2 \coloneqq$ `NaiveBranching`$(G - w, \ell)$
8 | return $\arg\max_{S \in \{S_1, S_2\}} |S|$

is not contained in at least ℓ triangles and we may thus remove vertices with degree at most $1/2 + \sqrt{1/4 + 2\ell}$ from the input graph. Furthermore, an edge e incident with a vertex v is contained in at most $\deg(v) - 1$ triangles. Thus, in case of Edge Triangle 2-Club we may remove vertices with degree at most ℓ.

Reduction Rule 7.2 (Low-Degree Rule (LDR))**.** *a) In case of* Vertex Triangle 2-Club*, remove all vertices that have degree at most* $1/2 + \sqrt{1/4 + 2\ell}$.

b) In case of Edge Triangle 2-Club*, remove all vertices that have degree at most* ℓ.

Observe that applying the LDR (Reduction Rule 7.2) may reduce the number of vertices and edges that need to be considered when applying the LTR (Reduction Rule 7.1). In the following, we denote the LDR and the LTR together as the *Basic Rules*. Note that these two rules were already observed by Almeida and Brás [3] for Vertex Triangle 2-Club. After applying the Basic Rules exhaustively, the input graph satisfies the triangle conditions posed on vertex-ℓ-triangle 2-clubs or edge-ℓ-triangle 2-clubs. However, the diameter might still be larger than 2. To handle this, we use a simple branching strategy which is based on the definition of incompatibility.

Definition 7.2. Two vertices u and w in a graph G are called *compatible* if and only if $\text{dist}_G(u, w) \leq 2$. Otherwise, u and w are called *incompatible*.

Let $G = (V, E)$ be an input graph where the triangle conditions are satisfied. If all vertices are pairwise compatible, one may return V as a solution. Otherwise, if there is a pair of incompatible vertices u and w, a solution cannot contain both

vertices at the same time. This gives rise to a simple search tree algorithm, where we branch into the cases where we either remove u or w from G. The pseudocode for this algorithm is given in Algorithm 7.1. Note that this simple branching algorithm also implies an FPT-algorithm for the dual parameter $n - k$.

Proposition 7.3. Vertex Triangle 2-Club *and* Edge Triangle 2-Club *can be solved in* $\mathcal{O}(2^{n-k}n^{\mathcal{O}(1)})$ *time.*

7.1.2 Marked Vertices

To speed up the branching, we use the idea of *marked vertices*, also used for other 2-Club variants [109, 145]. Intuitively, marked vertices must belong to the solution that we aim to compute. Suppose we know that the first recursive call in the naive branching algorithm did not result in an optimal solution. Then, removing u from G is not optimal and therefore, we may assume that u is part of the solution and mark u accordingly.

Formally, we extend the input of our problems to a graph $G = (V, E)$, an integer ℓ, and a set M of marked vertices and aim to find a solution S with $M \subseteq S$. We first describe how the concept of marked vertices can be used to improve the branching behind Algorithm 7.1. Afterwards, we provide further reduction rules that exploit marked vertices.

Branching with Marked Vertices. The idea is analogous to the idea behind Algorithm 7.1: If the triangle constraints are satisfied and all vertices are pairwise compatible, one may return V as a solution. Otherwise, we find two incompatible vertices u and w. Instead of branching into the two cases where either u or w is removed from the input graph, we branch into the two cases where u is removed from the input graph or u is marked.

We first present rules which guarantee that also in the second branch where u gets marked, the number of vertices is decreased by at least one. Recall that the marked vertex u is incompatible with at least one other vertex w. More generally, consider vertices that are incompatible with marked vertices. Since a marked vertex v must belong to a solution, no vertex x that is incompatible with v can be added to the solution. If x is unmarked, it can safely be removed from the input graph. Otherwise, if x is marked, there is no solution S containing both v and x and therefore, we may abort the branch.

Reduction Rule 7.3 (Incompatible-Resolution Rule (IRR))**.** *Remove all unmarked vertices that are incompatible to a marked vertex.*

Algorithm 7.2: `MarkedBranching` Algorithm.

Input: An instance $(G = (V, E), \ell)$ and a set of marked vertices $M \subseteq V$.
Output: A maximum-cardinality triangle 2-club S with $M \subseteq S$.

```
1 Apply Reduction Rules 7.2, 7.1, 7.3, 7.4, 7.5, and 7.6 on the input instance
2 if all v ∈ V are pairwise compatible, then
3 |  return V
4 else
5 |  Find an unmarked vertex u that is incompatible to at least one other
  |   vertex
6 |  S_1 ≔ MarkedBranching(G − u, ℓ, M)
7 |  S_2 ≔ MarkedBranching(G, ℓ, M ∪ {u})
8 |  return arg max_{S∈{S_1,S_2}} |S|
```

Reduction Rule 7.4 (Marked-Incompatible Rule (MIR))**.** *If an instance contains two incompatible vertices that are both marked, then return that there is no solution.*

The pseudocode for the branching with marked vertices is given in Algorithm 7.2. In Algorithm 7.2 we apply Reduction Rules 7.2, 7.1, 7.3, 7.4, 7.5, and 7.6 as follows: In one iteration we apply Reduction Rules 7.2, 7.1, 7.3, 7.4, 7.5, and 7.6 in this order. If at least one reduction rule applied, we start a new iteration.

Further Reduction Rules. The next rule deals with (marked) vertices that have a relatively small number of compatible vertices. Let (G, ℓ) be an instance of Vertex Triangle 2-Club or Edge Triangle 2-Club. Recall, that our algorithm stores the size k of a largest triangle 2-club detected so far. In other words, our algorithm now tries to find a triangle 2-club of size at least $k + 1$. Then, if a vertex v is compatible with at most $k - 1$ other vertices, the vertex v cannot be contained in a solution of size larger than k. If v is unmarked, then v can be removed from the input instance. Otherwise, if v is marked, we may abort this branch.

We present two variants of this rule. In the 2-NR (Reduction Rule 7.5) we directly count the number of compatible vertices of some vertex v, that is, we check if $N_2[v]$ has size at least k. In the LCR (Reduction Rule 7.6) we check if for each vertex $u \in V(G)$ whether u and v are compatible and we stop this procedure if the number of remaining vertices is not large enough such that v can have k compatible vertices.

Reduction Rule 7.5 (2-Neighborhood Rule (2-NR))**.** *Remove all vertices v with $|N_2[v]| \leq k$. If a marked vertex is removed, report that there is no solution of size larger than k.*

Reduction Rule 7.6 (Low-Compatibility Rule (LCR))**.** *Remove vertices whose number of compatible vertices is at most $k-1$. If a marked vertex is removed, report that there is no solution of size larger than k.*

Abusing notation, the refer to the rules LDR (Reduction Rule 7.2), LTR (Reduction Rule 7.1), 2-NR (Reduction Rule 7.5), and LCR (Reduction Rule 7.6) as the *Basic Rules*.

We describe two further rules to identify vertices that need to be marked based on the set of currently marked vertices.

Let v and u be vertices of the input graph. If v is contained in less than ℓ triangles after removing u, then u has to be contained in every solution that contains v. Consequently, if v is marked, we may also mark u. Since a solution of Edge Triangle 2-Club also fulfills the vertex-ℓ-triangle property, this also holds for the edge variant. This observation leads to the following reduction rule.

Reduction Rule 7.7 (Cascading Rule (CR))**.** *Let v be a marked vertex and let u be an unmarked vertex. Let x_v be the number of triangles which contain v, and let x_{uv} be the number of triangles which contain u and v. If $x_v - x_{uv} < \ell$, then mark also u.*

Suppose that two non-adjacent marked vertices u and w have exactly one common neighbor. Then, this unique common neighbor needs to be part of the solution S since otherwise the distance between u and w in $G[S]$ (in case of Vertex Triangle 2-Club) or in $(S, \widehat{E})$ (in case of Edge Triangle 2-Club) becomes larger than 2. Consequently, we may mark the unique common neighbor.

Reduction Rule 7.8 (No-Choice Rule (NCR))**.** *Let u and w be two marked vertices. If $uw \notin E(G)$ and both vertices have exactly one common neighbor v, then mark v.*

In the following we denote the IRR, MIR, CR, and NCR as the *Marking Rules*.

We refer to the algorithm variant that uses the Basic Rules and the Marking Rules and `Marked Branching` as the `Basic` version.

7.1.3 Conflict Graphs and Upper Bounds

We describe an upper bound of the solution size that is based on the conflict graph of the input graph. Given an instance $(G = (V, E), \ell)$, the *conflict graph of G* is a graph G_c on the same vertex set as G such that two vertices are adjacent in G_c if and only if they are incompatible in G.

Lemma 7.4. *The conflict graph G_c of a graph G can be constructed in $\mathcal{O}(nm)$ time. After deleting a vertex $w \in V$ in G, the conflict graph can be updated in $\mathcal{O}(\deg(w) \cdot m)$ time; after deleting an edge in G, the conflict graph can be updated in $\mathcal{O}(m)$ time.*

Proof. We may assume that the input graph is reduced with respect to the LDR (Reduction Rule 7.2) and therefore, G contains no isolated vertices.

The conflict graph can be constructed as follows: We start with an empty graph. For each vertex v we perform two BFS steps to compute $N_2[v]$ and add conflict edges vw to G_c for every $w \notin N_2[v]$. Consequently, G_c can be computed in $\mathcal{O}(nm)$ time.

We next consider the running time for updating G_c after deleting some vertex $w \in V$. To this end, note that removing w does not repair any incompatibility, but instead new incompatibilities may emerge, since two neighbors x and y of w may now be incompatible. To find these new incompatibilities, we perform a BFS from each neighbor of w. This can be done in $\mathcal{O}(\deg(w) \cdot m)$ time.

In case of deleting an edge xy, all new arising incompatibilities contain one of the vertices x or y. Thus, two calls of BFS are sufficient to update G_c accordingly in $\mathcal{O}(m)$ time. □

As described above, we assume that we know that a solution of size k exists and we aim to find a solution of size larger than k. Consider the following reduction rule, also used for other 2-Club variants [109, 145].

Reduction Rule 7.9 (Matching Rule)**.** *Compute the size b of a maximum matching for G_c. If $|V(G_c)| - b$ is at most k, then return no.*

Proposition 7.5. *Reduction Rule 7.9 is correct.*

Proof. Let M be a matching of size b. To prove the correctness of the rule, we show that every solution S has size at most $|V(G_c)| - b$.

Let S be a solution. Then, S does not contain two vertices that are adjacent in G_c and thus, S contains at most b vertices that are endpoints of edges in M. Therefore, S consists of at most $|V(G_c)| - b$ vertices. □

We refer to the algorithm variant that works like `Basic` but additionally uses the conflict graph and the Matching Rule (Reduction Rule 7.9) as `Basic+UB`.

7.2 Lower Bounds

To obtain a good initial solution we implemented two heuristics. These heuristics provide a lower bound for the size of the largest triangle 2-club. The first heuristic determines the largest triangle 2-club in the closed neighborhood of any vertex $v \in V(G)$ and the second one greedily determines a triangle 2-club in $N_2[v]$ for any $v \in V(G)$.

A Lower Bound Based on Closed Neighborhoods. We next describe the known *Neighborhood-Lower Bound (N-LB)* [3] and bound the worst-case running time for computing the lower bound. For each vertex $v \in V(G)$, we compute a largest triangle 2-club S_v that is contained in $N[v]$ and also contains v. To do this, we first construct the induced subgraph of $N[v]$, denoted G_v, and then apply Reduction Rule 7.1 to G_v. Note that S_v is a triangle 2-club: S_v fulfills the triangle property due to Reduction Rule 7.1 and S_v is a 2-club since v is universal in S_v. Moreover, since we search for a solution containing v we may abort if Reduction Rule 7.1 removes v. By S_v we denote the vertex set of the reduced graph. The heuristic then returns a vertex set S_v which has maximum size of any of the sets S_u, $u \in V(G)$. Next, we show that if a maximal triangle 2-club S has a universal vertex, then this heuristic will find S. In other words, in such a scenario the N-LB returns an optimal solution. For this, it is sufficient to show the following statement.

Proposition 7.6. *For each $v \in V(G)$ the N-LB returns a largest solution S_v that is contained in G_v and contains v.*

Proof. Let S be a largest triangle 2-club in G_v. We show that S_v contains all vertices of S.

First, we show the statement for the vertex-variant. If S_v is nonempty, then the vertex v is not returned by the LTR: v is universal and thus contained in a maximum number of triangles in G_v. Thus, S_v is a triangle 2-club. Now, assume towards a contradiction that S_v does not contain all vertices of S. Let w be the first vertex of S which gets deleted by the algorithm and consider the application of the LTR that deleted w. Since $w \in S$, we know that w is contained in at least ℓ triangles in G_v, all of these triangles are present when w is deleted, a contradiction to the definition of the LTR. Thus, such a vertex w does not exist and the algorithm returns S.

Second, we show the statement for the edge-variant. Let $G^* = (S_v, E^*)$ denote the graph at the end of the algorithm. We first show that S_v is an edge-ℓ-triangle 2-club in G. We show this by showing that G^* has diameter at most 2 and every edge of E^* is in at least ℓ triangles in G^*. The latter claim follows from the fact that the LTR has been applied exhaustively. It remains to show that G^* has diameter at most 2. To show this, we prove that in G^*, the vertex v is adjacent to all other vertices. Assume that the algorithm deletes an edge $\{v, u\}$ that is incident with v. Then all other edges $\{u, w\}$ incident with u will also be deleted by the algorithm: Since v is a universal vertex in G_v, for every triangle $\{u, w, x\}$ with $x \neq v$, the set $\{u, v, x\}$ is also triangle. Hence, the number of triangles containing $\{v, u\}$ is at least as large as the number of triangles containing $\{u, w\}$. As a consequence, whenever an edge $\{v, u\}$ is deleted, all edges incident with u are deleted. Consequently, u is deleted as well.

Altogether, v is adjacent to every other vertex of G^*. Thus, S_v is an edge-ℓ-triangle 2-club.

It remains to show that S_v contains all vertices of S. Assume towards a contradiction that this is not the case and let $(S, \widehat{E})$ denote the corresponding spanning subgraph. Now, assume towards a contradiction, that the algorithm deletes at least one edge of $\widehat{E}$. Let e be the first edge of $\widehat{E}$ which gets deleted, and consider the application of the LTR that deletes e. Since $e \in \widehat{E}$ we know that e is contained in at least ℓ triangles in $(S, \widehat{E})$. Since all edges of $\widehat{E}$ are present when e is deleted, it is contained in at least ℓ triangles at the time of its deletion, a contradiction to the definition of the LTR. Hence, such an edge e does not exist and thus G^* contains every edge of $\widehat{E}$ and therefore also all vertices of S. □

Next, we bound the overall running time of computing the N-LB.

Proposition 7.7. *For* Vertex Triangle 2-Club *and for* Edge Triangle 2-Club *the overall running time of the N-LB is* $\mathcal{O}(m \cdot d^2)$.

Proof. Fix some degeneracy ordering σ of G. For each vertex v, the graph G_v can be constructed in $\mathcal{O}(\deg(v) \cdot d)$ time by considering all neighbors w of v and then traversing the list containing all neighbors u of w that appear after w in σ. Thus, the total running time for computing all induced subgraphs is $\mathcal{O}(\sum_{v \in V(G)} \deg(v) \cdot d) = \mathcal{O}(m \cdot d)$. For each vertex v, the graph G_v is d-degenerate and thus has $\mathcal{O}(\deg(v) \cdot d)$ edges. By Lemma 7.1, the exhaustive application of the LTR on G_v thus takes $\mathcal{O}(\deg(v) \cdot d^2)$ time. Hence, the overall running time for the application of the rule for all graphs $v \in V(G)$ is $\mathcal{O}(\sum_{v \in V(G)} \deg(v) \cdot d^2) = \mathcal{O}(m \cdot d^2)$. □

In the worst-case, when $d = \Theta(n)$, the running time bound of the N-LB becomes $\mathcal{O}(n^4)$ and thus is impractical. In sparse real-world graphs, however, the degeneracy takes on very small values and the running time bound guarantees that the algorithm is fast. In particular, when the degeneracy is constant, we achieve a linear running time.

A Greedy Lower Bound. For each vertex $v \in V(G)$, we compute a triangle 2-club S_v that is contained in the closed second neighborhood $N_2[v]$ and contains v as follows: We first construct the induced subgraph $G_v \coloneqq G[N_2[v]]$ and apply the LTR (Reduction Rule 7.1) to G_v. By S_v we denote the vertex set of G_v after the application of the LTR (Reduction Rule 7.1). Each vertex in S_v or each edge with two endpoints in S_v is contained in at least ℓ triangles. However, S_v is not necessarily a 2-club. To solve this issue, we test whether S_v is a 2-club and return a vertex

pair $\{u, w\}$ of distance at least 3 in G_v. If $u = v$ or $w = v$, then we remove the other vertex since we aim to find the largest triangle 2-club containing v. Otherwise, we greedily remove the vertex of u and w which is in less triangles in G_v. Afterwards, we again apply the LTR (Reduction Rule 7.1). We continue with this procedure until the resulting vertex set S_v is a triangle 2-club. We call this the *Greedy-Lower Bound (G-LB)*. Next, we bound the overall running time of the G-LB.

Proposition 7.8. *For* Vertex Triangle 2-Club *and for* Edge Triangle 2-Club *the overall running time of the G-LB is* $\mathcal{O}(m \cdot \Delta^3 \cdot d^2)$.

Proof. The graph $G_v = G[N_2[v]]$ has less than $\deg(v) \cdot \Delta$ vertices and can be constructed in $\mathcal{O}(\deg(v) \cdot \Delta \cdot d)$ time using the degeneracy-ordering adjacency lists as detailed in the running time bound proof for the N-LB. The number of triangles in G_v is $\mathcal{O}(\deg(v) \cdot \Delta \cdot d^2)$ and again, these triangles can be computed in the corresponding running time. The total time for checking the triangle counters and deleting vertices or edges when their counters are too low is again upper-bounded by the number of triangles in G_v and thus bounded by $\mathcal{O}(\deg(v) \cdot \Delta \cdot d^2)$. The main running time bottleneck is the repeated check whether the current graph G_v has diameter 2 for which we make use of the conflict graph. By Lemma 7.4, the conflict graph can be constructed in $\mathcal{O}(\deg(v)^2 \cdot \Delta^2 d)$ time. Afterwards, we need to update the conflict graph after vertex and edge deletions. The running time for the update after deleting a vertex u is $\mathcal{O}(\deg(u) \cdot m)$, again by Lemma 7.4. The worst-case overall running for the updates thus is

$$\mathcal{O}\left(\sum_{u \in N_2[v]} \deg_v(u) \cdot \deg(v) \cdot \Delta \cdot d) = \mathcal{O}(\deg(v)^2 \cdot \Delta^2 \cdot d^2\right),$$

where $\deg_v(u)$ is the degree of u in G_v, if we essentially delete all vertices and edges of G_v either by reduction rules or by greedy choice. The total running time for computing the lower bounds for all vertices is $\mathcal{O}(\sum_{v \in V(G)} \deg(v)^2 \cdot \Delta^2 \cdot d^2) = \mathcal{O}(m \cdot \Delta^3 \cdot d^2)$. □

Again, on very dense graphs, the running time is impractical. While the running time becomes acceptable on sparse graphs with moderate maximum degree, it is still much higher than the running time for the N-LB. This can also be observed for our implementation, which prompted us to add several speed-ups to the implementation of the G-LB.

The algorithm variant that works like `Basic+UB` but additionally uses the N-LB is called `N-LB`. The variant that additionally uses both lower bounds is called `Multi-LB`.

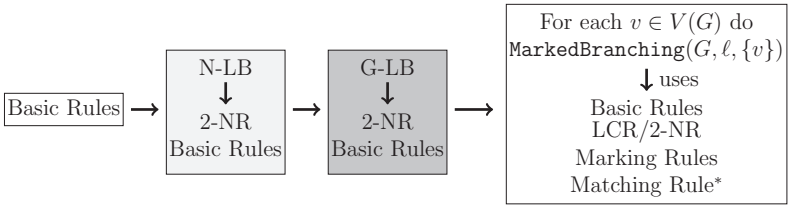

Figure 7.1: Sequence of our algorithm. Everything described in white boxes is done in each of the 4 variants. Everything within the light-gray box is done in the 2 variants using a lower bound (`N-LB` and `Multi-LB`). Furthermore, the gray box is used only for `Multi-LB`. The Matching Rule (marked with an asterisk (*)), is used in each variant except `Basic`.

7.3 Implementation Details

An outline of our algorithm is given in Figure 7.1. By k we denote the size of a largest triangle 2-club detected so far. Recall that the LCR (Reduction Rule 7.6) and the 2-NR (Reduction Rule 7.5) serve the same purpose; identifying vertices which have at most $k-1$ compatible vertices. The 2-NR is faster than the LCR if $N_2[v]$ is small. Hence, we have chosen to use the 2-NR for low densities. Another reason for our choice is that if the density of $N_2[v]$ is large, the LCR is not much slower than the 2-NR. Here, the *density* is the ratio between the edges present in a graph and the maximum number of edges that the graph can contain. In other words, on the basis of the density of $N_2[v]$ the algorithm decides whether it uses the 2-NR or the LCR in each subsequent call of `MarkedBranching`$(G, \ell, \{v\})$. We use the 2-NR if the density of $N_2[v]$ is at most 0.05; otherwise the LCR is used.

We use a global triangle list to allow for a fast check of all operations related to triangles, for example, whenever we compute the set of triangles containing one vertex u or an edge uw. For the G-LB and branching, we sort the vertices descendingly according to the size of their 2-neighborhood. Since the computation of the N-LB is fast, we use an arbitrary vertex ordering to compute the N-LB. After the branching on vertex v is completed, vertex v is removed from the graph.

Data Structures. Our algorithm assigns each vertex a unique ID. To represent a graph we use Hash Sets and Hash Maps. More precisely, for the vertex set we rely on a Hash Set to allow for a fast check whether a vertex is present. The set of edges

is organized in an adjacency list. More precisely, a Hash Map uses the vertex IDs to assign each vertex a distinct Hash Set, containing its neighbors, that is, the IDs of each adjacent vertex. Furthermore, we use a stack to reverse the operations in the search tree efficiently.

Compatibility Test and Conflict Graph. To test whether two vertices u and w are compatible, that is, have distance at most 2, we first check whether $u = w$. Afterwards, we check whether u and w are adjacent and then we check if u and w have a common neighbor. For any vertex v we create the conflict graph G_c of $N_2[v]$ by using the above mentioned compatibility test. We do *not* create the conflict graph of the complete input graph since this is too time consuming and needs too much memory. Whenever a vertex w is deleted from $N_2[v]$, we also delete w from G_c. Afterwards, we update G_c according to Lemma 7.4.

Reduction Rules. Next, we give some implementation details for reduction rules which are not implemented in a straightforward manner. Let $d_{\min}$ be the minimum degree of any vertex in a triangle 2-club. Recall that $d_{\min}$ is at most $1/2 + \sqrt{1/4 + 2\ell}$ for the vertex variant and that $d_{\min} = \ell + 1$ for the edge variant. In one application of the LDR (Reduction Rule 7.2), we first store all vertices of G that have degree smaller than $d_{\min}$. Afterwards, we remove all of these vertices. In one application of the LTR (Reduction Rule 7.1) we similarly store all vertices or edges which are in less than ℓ triangles, and then remove them simultaneously. In our implementations of the IRR (Reduction Rule 7.3), the MIR (Reduction Rule 7.4), and the LCR (Reduction Rule 7.6), we count the number of incompatibilities of $w \in N_2[v]$ and stop if $|N_2[v]| - k$ incompatibilities are found.

Lower Bounds. Before we compute the G-LB, we use the N-LB, to obtain an initial lower bound. Afterwards, the subsequent application of the Basic Rules and the 2-NR removes many further vertices from the graph and thus the G-LB has to be computed only for a fraction of the initial vertices. Since the running time for one iteration of the G-LB is much larger than for one iteration of the N-LB, this decreases the overall running time. Preliminary experiments showed that this is beneficial for small running times. In the following, we denote this combined lower bound by *N+G-LB*.

In both lower bounds, that is, in the N-LB and N+G-LB, we compute a lower bound for each vertex $v \in V(G)$. For this, we check if the size of $N[v]$, or $N_2[v]$, respectively, is larger than k, otherwise we abort and remove v from the graph. The removal of v is beneficial since after this deletion for another vertex w may now

also observe that $|N[w]| \leq k$, or $|N_2[w]| \leq k$, respectively, and hence we may also immediately abort and remove w from G. Next, we construct the subgraph induced by $N[v]$, or $N_2[v]$, respectively.

Afterwards, we use the LDR (Reduction Rule 7.2) as described above to remove vertices which cannot have sufficiently many triangles. Recall that in this iteration we search the largest triangle 2-club containing v. Thus, if the LDR (Reduction Rule 7.2) or other rules removes v, then we can abort this iteration.

In the N-LB we then apply the LTR (Reduction Rule 7.1) in the following way: Step 1: Apply LDR (Reduction Rule 7.2) exhaustively. Step 2: Perform one application of the LTR. If at least one vertex was deleted in this application, go back to Step 1.

In the N+G-LB we then construct the conflict graph (see Section 7.1.3). This allows us to use the LCR (Reduction Rule 7.6), that is, to remove vertices which cannot be part of a solution of at least $k+1$ vertices. Next, we establish the triangle property as follows:

1. We apply the LCR (Reduction Rule 7.6).

2. If the algorithm removed at least one vertex in Step 1, we apply the LDR (Reduction Rule 7.2) and go back to Step 1.

3. We perform one iteration of the LTR (Reduction Rule 7.1), that is, we remove all vertices/edges in less than ℓ triangles.

4. If the algorithm removed at least one vertex in Step 3, we apply the LDR (Reduction Rule 7.2) and go back to Step 1.

When the algorithm does not go back to Step 1 in Step 4, the triangle property is established. Now, if the number of the remaining vertices in $N_2[v]$ after the application of all these reduction rules is at most k, we delete v from the input graph. This is correct since until this point we only applied reduction rules and hence there is no solution of size at least $k+1$ containing v. With the removal of v of the input graph during the lower bound computation we avoid having to apply all these reduction rules for v again during branching.

Next, similar to Reduction Rule 7.9 we compute a maximum matching M of the conflict graph, by greedily adding edges to M. For each edge in M, we greedily remove the vertex which is in less triangles (vertex variant) or has smaller degree (edge variant). Note that if any edge of M contains v, then we always remove the other vertex because of the premise that we search for a solution containing v. Now,

we repeat the above procedure of establishing the triangle property and computing a matching until the remaining vertex set fulfills the triangle and the diameter property.

Our implementation of N+G-LB seems rather complicated, but preliminary experiments showed that this procedure is indeed necessary to achieve small running times.

7.4 Experiments

Each experiment was performed on a single thread of an Intel(R) Xeon(R) Silver 4116 CPU with 2.1 GHz, 24 CPUs and 128 GB RAM running Java openjdk 17.0.2. Our algorithms are implemented in Java. As benchmark data set we used 67 social, biological, and technical networks obtained from the Network Repository [200], KONECT [153], and the 10th DIMACS challenge [12]. These networks range from less than 100 vertices to up to 300 000 vertices. All networks except five are sparse with density at most 0.05. Roughly 10 networks have less than 100 vertices, 25 networks have between 100 and 1 000 vertices, 25 have between 1 000 and 10 000 vertices, and 10 have more than 10 000 vertices. Furthermore, we tested 27 different values for ℓ in the range of 1 to 100. More precisely, we tested our algorithm for each value in

$$\{1, 2, \ldots, 6, 7, 9, 11, 13, 15, 20, 25, \ldots, 90, 100\}.$$

For each instance, we set a time-out of 1 hour. The time needed to read the graph is not included in the running time. Our source code, the list of all networks used in our experiments, and our result files are available at `https://www.uni-marburg.de/en/fb12/research-groups/algorith/t2c.zip`. The running times for the ILP are the running times obtained by Almeida and Brás [3]. Almeida and Brás [3] implemented their ILP in CPLEX (`https://www.ibm.com/de-de/products/ilog-cplex-optimization-studio`). The specifications of the computer used by Almeida and Brás [3] are very similar to the specifications of our computer. Thus, the running times are comparable.

The performance of the four variants of our algorithm for Vertex Triangle 2-Club and $\ell \leq 5$ is shown in the left part of Figure 7.2. `Basic` is substantially slower than `Basic+UB` which in turn is substantially slower than `Multi-LB`. Furthermore, `N-LB` is even faster than `Multi-LB`. The performance of the four variants of our algorithm for Vertex Triangle 2-Club and $6 \leq \ell \leq 15$ is shown in the right part of Figure 7.2. All four variants are substantially faster for $6 \leq \ell \leq 15$, where the LDR (Reduction Rule 7.2) and the LTR (Reduction Rule 7.1) are applied more often in the initial data reduction which deletes significantly more vertices before the

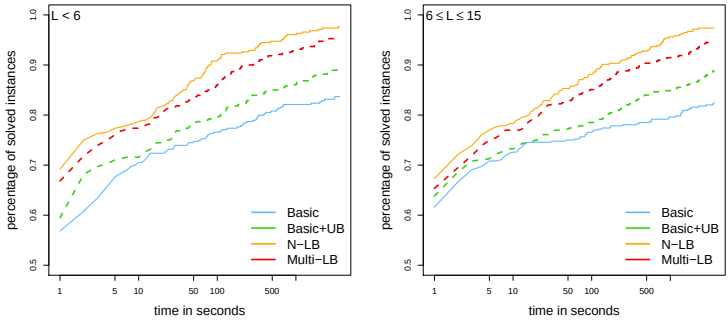

Figure 7.2: Comparison of the four variants of our algorithm for Vertex Triangle 2-Club for $\ell \leq 15$.

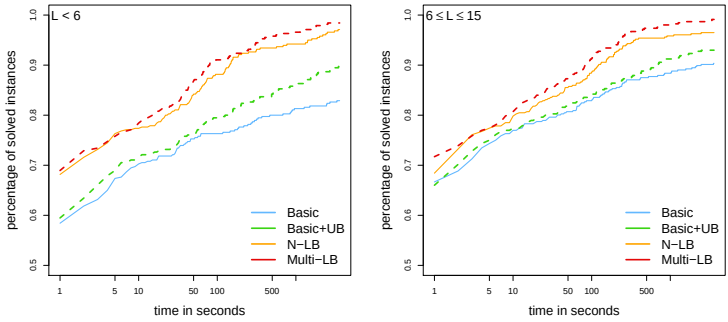

Figure 7.3: Comparison of the four variants of our algorithm for Edge Triangle 2-Club for $\ell \leq 15$.

algorithm computes a lower bound and applies branching.

The performance of the four variants of our algorithm for Edge Triangle 2-Club is shown in Figure 7.3. The left part shows our results for $\ell \leq 5$ and the right part shows our results for $6 \leq \ell \leq 15$. Similar to the vertex variant `Basic` is substantially slower than `Basic+UB` which in turn is substantially slower than `N-LB`.

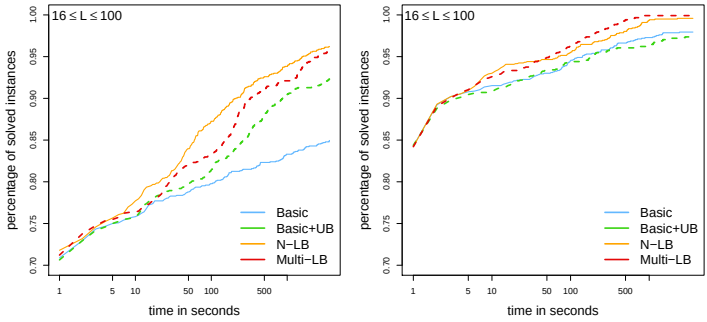

Figure 7.4: Comparison of the four variants of our algorithm for $\ell \geq 16$. The left plot shows our results for Vertex Triangle 2-Club and the right plot shows our results for Edge Triangle 2-Club.

Table 7.1: Average quality of both lower bounds for both the edge and the vertex variant for different values of ℓ.

LB	Vertex Variant			Edge Variant		
	$\ell \leq 5$	$6 \leq \ell \leq 15$	$\ell \geq 16$	$\ell \leq 5$	$6 \leq \ell \leq 15$	$\ell \geq 16$
N-LB	96.0%	93.8%	94.7%	97.1%	97.0%	97.4%
N+G-LB	99.9%	99.8%	99.5%	99.9%	99.8%	98.3%

In contrast to the vertex variant, `Multi-LB` is faster than `N-LB`. Again, all variants are substantially faster for larger ℓ.

The performance of the four variants for $\ell > 15$ of our algorithm for Vertex Triangle 2-Club and Edge Triangle 2-Club is shown in Figure 7.4. For Vertex Triangle 2-Club the results are similar to $\ell \leq 15$, that is, `Basic` is substantially slower than `Basic+UB` which in turn is substantially slower than `Multi-LB`, and `N-LB` is the fastest. For Edge Triangle 2-Club the result is different: `Basic` is faster than `Basic+UB` which is slower than `N-LB` and `Multi-LB` is the fastest. Again, all four variants are substantially faster for larger $\ell > 15$ than for $\ell \leq 15$ for both Vertex Triangle 2-Club and Edge Triangle 2-Club. The main reason for this result is that the LDR (Reduction Rule 7.2) and the LTR (Reduction Rule 7.1) are applied more often in the initial data reduction which removes a significant portion of the

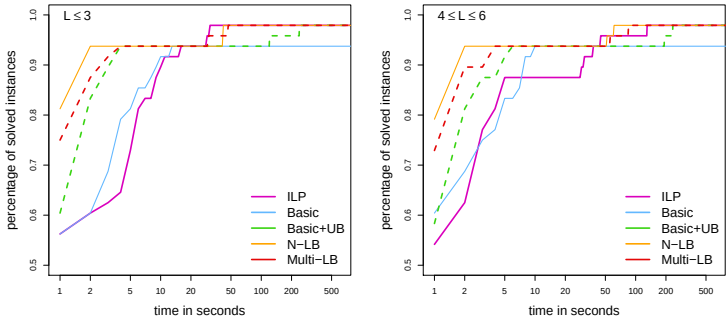

Figure 7.5: Comparison of the 4 variants of our algorithm for Vertex Triangle 2-Club with the ILP of Almeida and Brás.

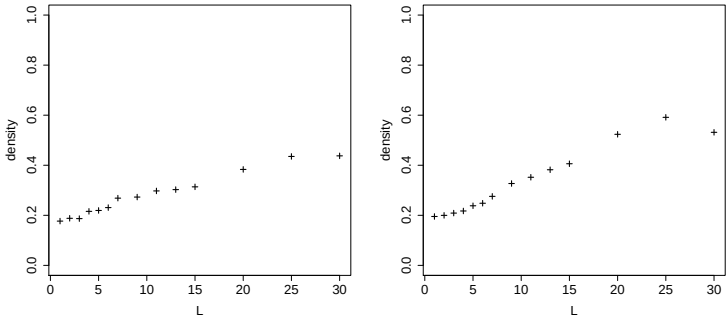

Figure 7.6: Dependence of the density on ℓ (L). The let plot shows the result for Vertex Triangle 2-Club and the right plot shows the result for Edge Triangle 2-Club.

vertices from the graph. This is especially the case for Edge Triangle 2-Club since initially all vertices with degree at most ℓ get removed.

Interestingly, `Multi-LB` was only beneficial in terms of running time for the edge-variant, for the vertex variant the running time increased compared with `N-LB`. However, such a behavior cannot be observed if we compare the size of the lower bounds

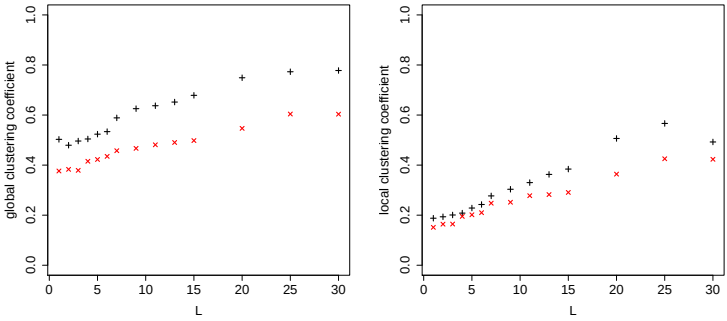

Figure 7.7: Dependence of the average global and the smallest local clustering coefficients on ℓ (L). Black +-signs show the values for the edge variant, red crosses show the values for the vertex variant.

N-LB (used in `N-LB`) and N+G-LB (used in `Multi-LB`) with the size of an optimal solution, see Table 7.1. For both the edge and the vertex variant the N+G-LB is much better than the N-LB. Surprisingly, the difference of the quality of these lower bounds is larger in the vertex variant than in the edge variant despite the fact the `Multi-LB` is only faster than the `N-LB` for the edge variant.

We compared our algorithm with the ILP formulation from Almeida and Brás [3] for Vertex Triangle 2-Club in Figure 7.5. Since the source code of the ILP was not available to us, we compared our algorithm with the ILP only on the instances used by Almeida and Brás [3]. The ILP formulation was only faster for the graph *polblogs*. Note that all instances of the graph *polblogs* were solved by our algorithm by only applying data reduction rules. For all remaining instances our algorithm outperformed the ILP.

We also considered the impact of ℓ on the density, see Figure 7.6. The figure shows the average densities for all instances that were solved within the time limit and had non-empty solutions for all $\ell \leq 30$. In general, the density grows with ℓ. Already for $\ell = 1$, the density is relatively high. One can see that the density increases more strongly with increasing ℓ in the edge variant than in the vertex variant. The main reason for this is that the minimum degree in an edge-ℓ-triangle 2-club is higher than the minimum degree in a vertex-ℓ-triangle 2-club, especially for $\ell \geq 10$.

Finally, we consider the impact of ℓ on global and local clustering coefficients,

see Figure 7.7. The figure shows the average coefficients for all instances that were solved within the time limit and had non-empty solutions for all $\ell \leq 30$. In general, both coefficients grow with ℓ. As expected, the growth is more rapid for the stricter edge-variant where the minimum degree is $\ell + 1$. Observe that already for $\ell = 1$, the global clustering coefficient is relatively high for both variants. Naturally, the smallest local clustering is comparably smaller but achieves high values for $\ell \leq 10$ in both variants. Summarizing, this shows that the computed solutions fulfill further desirable community properties.

7.5 Conclusion

We provided an exact branch-and-bound solver for Vertex Triangle 2-Club and for Edge Triangle 2-Club. We showed that our solver outperforms an existing ILP by Almeida and Brás [3] for Vertex Triangle 2-Club. Furthermore, we showed that the local and global clustering coefficient in a triangle 2-club in real-world instances is much higher than the guaranteed local or global clustering coefficient by the definition of this model (see [3] and Chapter 6). Also, our experiments showed that both the local and global clustering coefficient is higher in the edge variant than the corresponding values in the vertex variant. Since also edge-ℓ-triangle 2-clubs can be found faster than vertex ℓ-triangle 2-clubs (which is mainly because of the higher minimum degree in the edge variant), we conclude that the edge-ℓ-triangle 2-club model is not only preferable from a modeling point of view (an edge-ℓ-triangle 2-club is robust against up to ℓ edge deletions, see Proposition 6.4), but also regarding the running time and quality of the solution.

Our naive branching algorithm (Algorithm 7.1) implied that Vertex Triangle 2-Club and Edge Triangle 2-Club can be solved in $\mathcal{O}(2^{n-k}n^{\mathcal{O}(1)})$ time (Proposition 7.3). This result can be extended to show that Vertex Triangle s-Club and Edge Triangle s-Club for each s can be solved in $\mathcal{O}(2^{n-k}n^{\mathcal{O}(1)})$ time by adapting the definition of incompatible vertices (distance $s+1$ instead of 3, see Definition 7.2).

Table 7.1 shows that our lower bounds, especially the N+G-LB, are very close to an optimal solution and often already find an optimal solution. As a side result, we showed that the running time of the G-LB is $\mathcal{O}(m \cdot \Delta^3 \cdot d^2)$ (Proposition 7.8). This running time is very high, even on sparse real-world graphs. This high running time was our motivation to speed-up the computation of the G-LB by using the N-LB beforehand and to use the conflict graph and the reduction rules like the LDR, LTR, and LCR (see Section 7.3). All these speed-ups made the usage of the G-LB practical. Nevertheless, the running time of the N+G-LB is still quite high. Hence, a next step would be to find more reduction rules or weaker but faster versions of

existing reduction rules to accelerate the N+G-LB. Such tricks or reduction rules are not only desirable for the N+G-LB, but also for the branching: The number of nodes in the search tree is relatively low compared to the overall running time. Hence, the time spent in one search tree node is quite high. The main reason for this is that in each node we have to check if the corresponding graph is a triangle 2-club. This can be observed, for example, in the instances in which the lower bound equals the size of an optimal solution since our experiments showed that then the search tree very often has depth one.

Another possibility is to use different heuristics in the computation of the lower bounds: Currently if a pair of incompatible vertices u and w is detected, we delete the vertex which is in less triangles (vertex variant) or which has lower degree (edge variant). Hence, it is interesting to study different greedy criteria according to which a vertex of a pair of incompatible vertices is removed. One option is to remove the vertex with the lowest number of incompatibilities. Note that this can be checked efficiently because we use the conflict graph in the G-LB. More precisely, the number of incompatibilities of vertex v is exactly the degree of v in the conflict graph. This idea will not improve our lower bounds since they are already very close to optimal solutions. However, it may reduce the time which is necessary to compute the lower bounds since this different heuristic might lead to more subsequent applications of the LTR (Reduction Rule 7.1).

Another way to improve our algorithm is to extend the definition of being incompatible. Currently, two vertices are incompatible if and only if their distance is at least 3. Hence, this definition only uses the 2-club property, but not the triangle property. In the following, we describe one way to incorporate the triangle property into the definition of incompatibility. Roughly speaking, two vertices u and w are also incompatible if too many vertices in a triangle with u have only few compatible vertices in triangles with w. Let us describe this idea more precisely. Let T_u be the set of vertices which are in a triangle together with vertex u. Now, two vertices u and w are incompatible if the following algorithm returns no: Compute the sets T_u and T_w. While there exists a vertex $v \in T_u$ which has distance at most 2 to less than ℓ vertices in T_w, delete v from T_u and do the same with T_w. Return "no" as soon as $|T_u| < \ell$ or $|T_w| < \ell$ or if a marked vertex gets removed.

It would be interesting to also develop ILP formulations for Edge Triangle s-Club and compare them with our algorithm. It seems very promising to tune such an ILP with the LDR (Reduction Rule 7.2) and the LTR (Reduction Rule 7.1) as an initial pre-processing as it was done by Almeida and Brás for Vertex Triangle s-Club. Furthermore, it is interesting to see whether providing the ILP for Edge Triangle s-Club with the lower bound of the N-LB decreases the running

time. The same test could then be done for the N+G-LB for the ILPs for Vertex Triangle s-Club and Edge Triangle s-Club.

Finally, it would be interesting to lift our implementation to be able to also solve Vertex Triangle s-Club and Edge Triangle s-Club for $s \geq 3$. Some reduction rules like the LTR (Reduction Rule 7.1) or the LDR (Reduction Rule 7.2) can be used directly for these problems. To use the LCR, one simply needs to adapt the definition of incompatibility. But not every reduction rule can be adapted that easily: For example the NCR (Reduction Rule 7.8) is not true anymore since two non-adjacent marked vertices u and w do not need to have a common neighbor anymore since $s \geq 3$. Another challenge is to develop good heuristics: For $s \geq 3$, a lower bound based only on the neighborhood of a vertex might have a too large difference to the value of an optimal solution. Finally, one bottleneck in our solver for $s = 2$ is the check whether a vertex set is a 2-club. For $s \geq 3$ this check needs even more time since it is not sufficient anymore to check whether each pair of vertices is adjacent or has a common neighbor. Hence, it is important to check the s-club property sufficiently fast.

Chapter 8

Covering Many (or Few) Edges with k Vertices in Sparse Graphs

In many fixed-cardinality optimization problems, the value of the objective function f does not only depend on the structure of the subgraph induced by the solution S [38, 66]. In many examples f depends on the number of edges that have one or two endpoints in S: In Maximum Partial Vertex Cover (MaxPVC), we require that at least t edges have at least one endpoint in S. MaxPVC has applications in facility location [85]. Conversely, in Minimum Partial Vertex Cover (MinPVC) we require that at most t edges have at least one endpoint in S. MinPVC is used to model the loading of semi-conductor components to be assembled into products [92]. In Max $(k,n-k)$-Cut, we require that at least t edges have exactly one endpoint in S. Max $(k,n-k)$-Cut is used as a group centrality measure where a group with many connections to the remaining graph is sought [73]. Furthermore, it is used to search for a ground state in the anti-ferromagnetic k-state Potts mode [226]. Furthermore, in Min $(k,n-k)$-Cut we require that at most t edges have exactly one endpoint in S. Min $(k,n-k)$-Cut is used in VLSI-design [155].

Also, some problems in which the objective function depends only on the structure of $G[S]$ fit into this setting: In the decision version of Densest k-Subgraph we require that there are at least t edges with both endpoints in S. Densest k-Subgraph has applications in spam detection [91] and in community mining [59]. Conversely, in Sparsest k-Subgraph we require that at most t edges have both endpoints in S. Sparsest k-Subgraph is used in tools to visualize stock market interactions [26].

All these above-mentioned problems have been studied extensively, for example in terms of their classic complexity, parameterized complexity and in terms of ap-

proximation. For example, Densest k-Subgraph as been shown to be NP-hard even if the maximum degree is 3 and the degeneracy is 2 [74]. Moreover, Densest k-Subgraph is W[1]-hard with respect to $k + t$ [62] since $t = \binom{k}{2}$ yields the Clique problem. Also, Densest k-Subgraph remains W[1]-hard parameterized by k even on graphs with degeneracy 2 and closure number 2 [198]. Furthermore, the problem also remains W[1]-hard with respect to the dual parameter $n - k$ [38]. A randomized FPT-algorithm for Densest k-Subgraph parameterized by $k + \Delta$ with running time $2^{\mathcal{O}(\Delta k)}$ was developed by Cai, Chan, and Chan [39]. Later, a derandomized algorithm with better worst-case running time of $\mathcal{O}(\Delta^k)$ was presented by Komusiewicz and Sorge [150]. Also, approximation algorithms for Densest k-Subgraph have been studied extensively [20, 74, 130]. An exact exponential-time algorithm with running time $\mathcal{O}(1.7315^n)$ was presented by Chang et al. [42].

Furthermore, Sparsest k-Subgraph is NP-hard [87, 126] and W[1]-hard with respect to $k + t$ [62] since $t = 0$ yields the Independent Set problem. Also, W[1]-hardness for $k+t$ can be shown for regular graphs [38] and for the dual parameter $n - k$ [38]. Cai, Chan, and Chan presented an FPT-algorithm for the parameter $k+\Delta$ [39]. Sparsest k-Subgraph is inapproximable for t with any factor since Independent Set is NP-hard and each independent set has 0 inner edges. Hence, approximation algorithms on graph classes in which Independent Set is solvable in polynomial time have been studied [224]. An exact exponential-time algorithm with running time $\mathcal{O}(1.7315^n)$ was presented by Chang et al. [42].

MaxPVC is NP-hard [87, 126] and W[1]-hard for k [38, 101] and $n - k$ [38, 62]. Furthermore, in contrast to Densest k-Subgraph and Sparest k-Subgraph, it admits an FPT-algorithm for t [22, 133]. An FPT-algorithm with respect to $\Delta + k$ was provided by Raman and Saurabh [198]. Amini et al. [8] improved this result to an FPT-algorithm for the parameter $d + k$ with running time $\mathcal{O}((dk)^k)$. Recently, this result was improved by Panolan and Yaghoubizade to an algorithm with running time $2^{\mathcal{O}(dk)}$ [187]. Furthermore, a factor 2-approximation was presented by Bshouty and Burroughs [36]. Later, this ratio was improved by a factor depending on the maximum degree [85, 104]. Also, genetic algorithms to solve the problem have been investigated [235].

MinPVC is NP-hard [87, 126]. Furthermore, MinPVC is also W[1]-hard with respect to k [38, 101] and also with respect to the dual parameter $n - k$ [38, 62]. Bonnet et al. presented an FPT-algorithm for the parameter $k+\Delta$ [29]. Furthermore, MinPVC admits an FPT-algorithm for parameter t [133]. There exists a factor 2-approximation and this is tight: under standard assumptions no approximation with factor $(1 - \epsilon)$ for any $\epsilon > 0$ is possible [86]. Furthermore, also FPT-approximations are studied [102, 164].

Max $(k, n-k)$-Cut is NP-hard [87, 126] and W[1]-hard with respect to the standard parameter solution size k [38, 60]. Furthermore, W[1]-hardness was also provided for the dual parameter $n-k$ [38]. The problem is FPT with respect to the parameter $\Delta + k$ [29]. Also, for parameter t an FPT-algorithm was presented by Bonnet et al. [29]. This algorithm was later improved by Saurabh and Zehavi [204]. Also, approximation algorithms for this problem were studied [1, 76].

Min $(k, n-k)$-Cut is NP-hard [87, 126] and W[1]-hard with respect to k and also for the dual parameter $n-k$ [38]. For parameterization with $\Delta + k$ Bonnet et al. [29] provided an FPT-algorithm, which was then improved by Saurabh and Zehavi [204]. Furthermore, also for parameter t an FPT-algorithm was provided by Cygan et al. [55]. A randomized approximation algorithm with ratio $(1+\varepsilon k)/\log(n)$ for some fixed $\varepsilon > 0$ was presented by Feige, Krauthgamer, and Nissim [75]. This ratio was later improved to $\mathcal{O}(\log(n))$ by Zhang [234].

We study the following general problem first defined by Bonnet et al. [29] that contains all of the above problems as special case.[1]

Max α-Fixed Cardinality Graph Partitioning (Max α-FCGP)

Input: A graph G, $k \in \mathbb{N}$, and $t \in \mathbb{Q}$.

Question: Is there a set S of exactly k vertices such that

$$\mathrm{val}(S) \coloneqq (1-\alpha) \cdot m(S) + \alpha \cdot m(S, V(G) \setminus S) \geq t \ ?$$

Here, $\alpha \in [0, 1]$, and $m(S)$ denotes the number of edges with two endpoints in S and $m(S, V(G) \setminus S)$ denotes the number of edges with exactly one endpoint in S. Naturally, one may also consider the minimization problem, denoted as Min α-Fixed Cardinality Graph Partitioning (Min α-FCGP), where we are looking for a set S such that $\mathrm{val}(S) \leq t$.

Furthermore, the value of α describes how strongly edges with exactly one endpoint in S influence the value of S relative to edges with two endpoints in S. For $\alpha = 1/3$, edges with two endpoints in S count twice as much as edges with one endpoint in S and, thus, every vertex contributes exactly its degree to the value of S. Hence, in this case, we simply want to find a vertex set with a largest or smallest degree sum and thus the problems Max α-FCGP and Min α-FCGP can be solved in polynomial time if $\alpha = 1/3$.

[1] On the face of it, the definition of Bonnet et al. [29] seems to be more general as it has separate weight parameters for the internal and outgoing edges. It can be reduced to our formulation by adapting the value of t and thus our results also hold for this formulation.

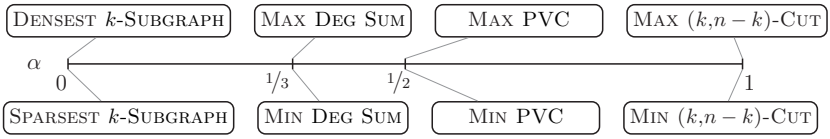

Figure 8.1: Problem definition cheat sheet.

More importantly, Max α-FCGP and Min α-FCGP contain all of the above-mentioned problems as special cases (see Figure 8.1). For example, Partial Vertex Cover (MaxPVC) is Max α-FCGP with $\alpha = 1/2$ as all edges with at least one endpoint in S count the same. Max $(k, n-k)$-Cut is Max α-FCGP with $\alpha = 1$ and Min $(k, n-k)$-Cut is Min α-FCGP with $\alpha = 1$ since in both cases edges with both endpoints in S are ignored. Sparsest k-Subgraph is Min α-FCGP with $\alpha = 0$ and Densest k-Subgraph is Max α-FCGP with $\alpha = 0$ as only the edges with both endpoints in S count. As discussed above, there exist values of α such that Max α-FCGP and Min α-FCGP are NP-hard and W[1]-hard on general graphs with respect to the natural parameter k [38, 53, 63, 101]. This hardness makes it interesting to study these problems on input graphs with special structure and Bonnet et al. [29] and Shachnai and Zehavi [209] studied this problem on bounded-degree graphs.

We continue this line of research and give a complete picture of the parameterized complexity of Min α-FCGP and Max α-FCGP on several types of sparse graphs that are described by structural parameters. In particular, we provide kernelization algorithms and kernel lower bounds for these problems, see Figure 8.2 for an overview.

Our results. We provide a complete picture of the parameterized complexity of Max α-FCGP and Min α-FCGP for all α with respect to the combination of k and five parameters describing the graph structure: the maximum degree Δ of G, the h-index of G, the degeneracy of G, the c-closure of G, and the vertex cover number vc of G. With the exception of the c-closure, all parameters are sparseness measures. The c-closure, first described by Fox et al. [84], measures how strongly a graph adheres to the triadic closure principle. Informally, the closure of a graph is small whenever all vertices with many common neighbors are also neighbors of each other. For a formal definition of all parameters refer to Section 2.1.

Our results are summarized by Figure 8.2. On a very general level, our main finding suggests that the degrading problems are much more amenable to FPT-

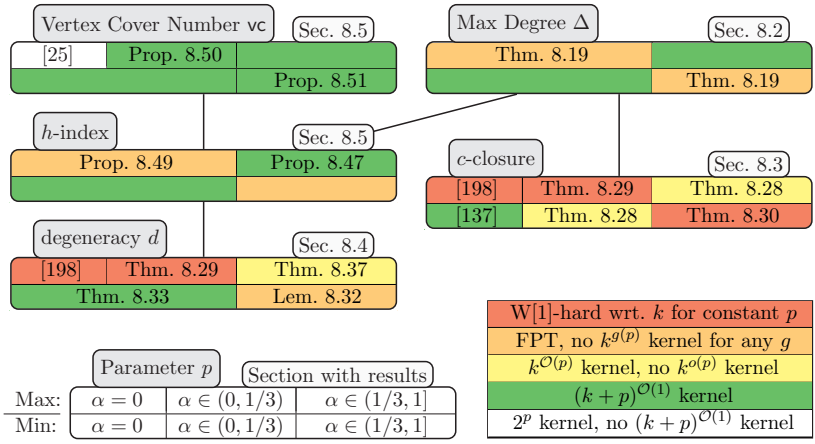

Figure 8.2: Overview over our results. Each box displays the parameterized results (see also bottom right) with respect to k and the corresponding parameter p for all variants (maximization, minimization, and all $\alpha \in [0, 1]$, see bottom left). Note that the split of the boxes is not proportional to the corresponding values of α. See Section 2.1 for the definitions of the parameters. A line from a box for parameter p to a box *above* for parameter p' implies that $p \in \mathcal{O}(p')$ on all graphs. Thus, hardness results hold also for all parameters below and tractability results for all parameters above.

algorithms and kernelizations than their non-degrading counterparts. No such difference is observed when considering the running time of FPT-algorithms for the parameter $k + \Delta$ but it becomes striking in the context of kernelization and when using secondary parameters that are smaller than Δ. Given the importance of the distinction between the degrading and non-degrading cases, we distinguish these subcases of Max α-FCGP and Min α-FCGP by name (Degrading Max α-FCGP, Non-Degrading Max α-FCGP, Degrading Min α-FCGP, Non-Degrading Min α-FCGP).

Independent of our work, a polynomial *compression* for MaxPVC (the special case of Max α-FCGP with $\alpha = 1/2$) of size $(dk)^{\mathcal{O}(d)}$ was recently discovered by Panolan and Yaghoubizade [187]. This result, together with our kernel of size $k^{\mathcal{O}(d)}$ (Theorem 8.37), answers an open question of Amini, Fomin, and Saurabh [8]. They asked whether MaxPVC admits a polynomial kernel in planar graphs. Note that

planar graphs have degeneracy 5.

On a technical level, by introducing an annotated version of the problem that keeps track of removed vertices, we separate and unify arguments that deal with vertices identified as (not) being part of a solution. In particular, we show that by introducing vertex weights (called counter) we can deal with vertices whose contribution is substantially below or above the average contribution that is necessary to reach the threshold t. More precisely, if the contribution of a vertex v is much above t/k, then we can add v to the solution and if it is much below t/k, then we can remove v. As a consequence, we can show that the weights can be bounded in the maximum degree of the annotated instance. This gives the kernels for the parameter $k + \Delta$.

The main step in the more sophisticated kernelizations for the degeneracy d and the c-closure is now to decrease the maximum degree of the instance as this allows us to use the kernel for $k + \Delta$. To decrease the maximum degree, for these parameters, we make use of Ramsey bounds. More precisely, the Ramsey bounds help to find a large independent set I such that all vertices outside of I have only a bounded number of neighbors in I. This then allows to prove by pigeonholing the following for the vertex v of I with the currently worst contribution to the objective function: No matter what the optimal solution selects outside of I, there is always some vertex of $I \setminus \{v\}$ that gives at least as good a contribution to the final solution as v. For the parameter c, we also need an additional pigeonhole argument excluding large cliques in order to apply the Ramsey bound. For the parameter d, we establish a new constructive Ramsey bound for $K_{i,j}$-free graphs that may be of independent interest.

We remark that when we describe the kernel size for $\alpha > 0$ (for instance, Proposition 8.11), the factor $1/\alpha$ is hidden in the $\mathcal{O}$ notation. We would like to emphasize, however, that the exponents in the kernel size such as $\mathcal{O}(c)$ and $\mathcal{O}(d)$ do not depend on $1/\alpha$. On the other hand, the lower bounds such as Theorem 8.19 hold indeed for all α in the range corresponding to the case.

We believe that this general approach could be useful for other parameterizations that are not considered in this work. A somewhat surprising consequence of our kernelizations is that Partial Vertex Cover and Max $(k, n-k)$-Cut not only behave in the same way but that the kernels for both problems can also be obtained by the same algorithms.

8.1 A Data Reduction Framework via Annotation

In this section, we introduce an annotated variant which gives more options for encoding information in the instances, allowing easier handling for kernelization. Moreover, to avoid repeating certain basic arguments, we provide general data reduction rules and statements used in the subsequent sections. Finally, we describe how to reduce from the annotated to the non-annotated problem variants in polynomial time.

In the annotated problem variant we encode that some vertices are decided to be in the solution and some vertices are decided to not be in the solution. To this end, we have additionally as input a (possibly empty) partial solution $T \subseteq V(G)$. Moreover, for each vertex we store a number $\mathsf{counter}\colon V \to \mathbb{N}$ which encodes the number of deleted neighbors not in the solution. We will assume throughout the paper that $\mathsf{counter}(v) = 0$ for every $v \in T$. For a set $S \subseteq V(G)$, we set

- $\mathsf{counter}(S) \coloneqq \sum_{v \in S} \mathsf{counter}(v)$ and
- $\mathrm{val}_G(S) \coloneqq \alpha(m(S, V(G) \setminus S) + \mathsf{counter}(S)) + (1-\alpha)m(S)$.

For a vertex $v \in S$, we set $\deg^{+\mathsf{c}}(v) \coloneqq \deg(v) + \mathsf{counter}(v)$.

Annotated Max α-FCGP
Input: A graph G, $T \subseteq V(G)$, $\mathsf{counter}\colon V(G) \to \mathbb{N}$, $k \in \mathbb{N}$, and $t \in \mathbb{Q}$.
Question: Is there a vertex set S of size k such that $T \subseteq S \subseteq V(G)$ and $\mathrm{val}_G(S) \geq t$ (Max) or $\mathrm{val}_G(S) \leq t$ (Min), respectively?

A vertex set S fulfilling these requirements is referred to as a *solution*. Now, we define the *contribution* of a vertex. The contribution of a vertex v is a measure on how much the value of a partial solution T increases if v is added to T. Note that our definition slightly differs from that of Bonnet et al. [29].

Definition 8.1. For a vertex set $T \subseteq V(G)$, we define the *contribution* of a vertex v as

$$\begin{aligned}\mathrm{cont}(v,T) &\coloneqq \alpha \cdot (|N(v) \setminus T| + \mathsf{counter}(v)) + (1-2\alpha)|N(v) \cap T| \\ &= \alpha \deg^{+\mathsf{c}}(v) + (1-3\alpha)|N(v) \cap T|.\end{aligned}$$

The contribution is chosen so that the value $\mathrm{val}(S)$ of a vertex set S computes as follows.

Lemma 8.2. *Let G be a graph and $S \coloneqq \{v_1, \ldots, v_\ell\} \subseteq V(G)$ a vertex set. Then, it holds that* $\mathrm{val}(S) = \sum_{i \in [\ell]} \mathrm{cont}(v_i, \{v_1, \ldots, v_{i-1}\})$.

Proof. Let $S_i = \{v_1, \ldots, v_{i-1}\}$ and $\overline{S_i} = \{v_i, \ldots, v_\ell\}$ for each $i \in [\ell]$. Observe that

$$m(S, V(G) \setminus S) = \sum_{i \in [\ell]} |N(v_i) \setminus S| = \sum_{i \in [\ell]} |N(v_i) \setminus S_{i-1}| - |N(v_i) \cap \overline{S_i}| \text{ and}$$
$$m(S) = \sum_{i \in [\ell]} |N(v_i) \cap S_{i-1}| = \sum_{i \in [\ell]} |N(v_i) \cap \overline{S_i}|.$$

We thus have

$$\begin{aligned}
\mathrm{val}(S) &= \alpha(m(S, V(G) \setminus S) + m(S) + \mathsf{counter}(S)) + (1 - 2\alpha) \cdot m(S) \\
&= \sum_{i \in [\ell]} \Big(\alpha[(|N(v_i) \setminus S_{i-1}| - |N(v_i) \cap \overline{S_i}|) + |N(v_i) \cap \overline{S_i}| + \mathsf{counter}(v)] \\
&\qquad + (1 - 2\alpha)|N(v_i) \cap S_{i-1}| \Big) \\
&= \sum_{i \in [\ell]} \mathrm{cont}(v_i, \overline{S_i}) = \sum_{i \in [\ell]} \mathrm{cont}(v_i, S_i),
\end{aligned}$$

which proves the lemma. □

For a vertex v and two sets $X \subseteq Y \subseteq V(G)$, we have $\mathrm{cont}(v, X) \geq \mathrm{cont}(v, Y)$ for $\alpha \in (1/3, 1]$ and $\mathrm{cont}(v, X) \leq \mathrm{cont}(v, Y)$ for $\alpha \in [0, 1/3)$ by Definition 8.1. Note that a function $f : 2^P \to \mathbb{Q}$ such that for each $X, Y \subseteq P$ with $X \subseteq Y$ and for each element $v \in P \setminus Y$ it holds that $f(X \cup \{v\}) - f(X) \geq f(Y \cup \{v\}) - f(Y)$ is called *submodular* and a function $g : 2^P \to \mathbb{Q}$ such that for each $X, Y \subseteq P$ with $X \subseteq Y$ and for each element $v \in P \setminus Y$ it holds that $g(X \cup \{v\}) - g(X) \geq g(Y \cup \{v\}) - g(Y)$ is called *supermodular*. By Lemma 8.2 we have $\mathrm{val}_G(X \cup \{v\}) = \mathrm{val}_G(X) + \mathrm{cont}(v, X)$. Hence, we conclude the following.

Observation 8.3. *The function* $\mathrm{val}_G(\cdot)$ *is submodular for* $\alpha \in (1/3, 1]$ *and supermodular for* $\alpha \in [0, 1/3)$.

8.1.1 Main Reduction Rules & Basic Exchange Argument

Annotations are helpful for data reductions in the following way: If we identify a vertex v that is (or is not) in a solution, then, we can simplify the instance as follows using the annotations.

Reduction Rule 8.1 (Inclusion Rule)**.** *If there is a solution* S *with* $v \in S \setminus T$*, then add* v *to* T*. If there is a vertex* $v \in T$ *with* $\mathsf{counter}(v) > 0$*, then decrease* t *by* $\alpha \cdot \mathsf{counter}(v)$ *and set* $\mathsf{counter}(v) \coloneqq 0$.

Reduction Rule 8.2 (Exclusion Rule)**.** *If there is a solution S with $v \notin S$, then for each $u \in N(v)$ increase* $\mathsf{counter}(u)$ *by one and remove v from G.*

The correctness of these two rules follows by the definitions of a partial solution and of the counter. Note that we maintain the aforementioned invariant that every vertex $v \in T$ has $\mathsf{counter}(v) = 0$ when applying the Inclusion Rule (Reduction Rule 8.1). Furthermore, we observe the following.

Observation 8.4. *The Inclusion Rule (Reduction Rule 8.1) and the Exclusion Rule (Reduction Rule 8.2) do not increase the parameters maximum degree Δ, c-closure, degeneracy d, vertex cover number* vc*, and h-index.*

The reduction rules themselves are simple. The difficulty lies of course in identifying vertices that are included in or excluded from some solution. In the respective arguments, we use the following notion of better vertices.

8.1.2 Better Vertices

The following notion captures a situation that frequently appears in our arguments for the annotated problem variant and allows for simple exchange arguments (see Lemma 8.6).

Definition 8.5. A vertex $v \in V(G)$ is *better* than $u \in V(G)$ with respect to a vertex set $T \subseteq V(G)$ if $\mathrm{cont}(v, T) \geq \mathrm{cont}(u, T)$ for the maximization variant (if $\mathrm{cont}(v, T) \leq \mathrm{cont}(u, T)$ for the minimization variant).

A vertex $v \in V(G)$ is *strictly better* than $u \in V(G)$ if for all $T \subseteq V(G)$ of size at most k we have $\mathrm{cont}(v, T) \geq \mathrm{cont}(u, T)$ for the maximization variant ($\mathrm{cont}(v, T) \leq \mathrm{cont}(u, T)$ for the minimization variant).

When we simply say that v is better than u, we mean that v is better than u with respect to the empty set. The following lemma immediately follows from Lemma 8.2.

Lemma 8.6. *Let S be a solution of an instance of* Annotated α-FCGP*. Suppose that there are two vertices $v \in S$ and $v' \notin S$ such that v' is better than v with respect to $S \setminus \{v\}$ or v' is strictly better than v. Then, $S' \coloneqq (S \setminus \{v\}) \cup \{v'\}$ is also a solution.*

Proof. We give a proof for the maximization variant; the minimization variant follows analogously. By Lemma 8.2, we have $\mathrm{val}(S') = \mathrm{val}(S \setminus \{v\}) + \mathrm{cont}(v', S \setminus \{v\}) \geq \mathrm{val}(S \setminus \{v\}) + \mathrm{cont}(v, S \setminus \{v\}) = \mathrm{val}(S)$. Here, the inequality follows from the fact that v' is better than v. □

Observe that the contribution of any vertex v differs from $\alpha \deg^{+c}(v)$ by at most $|(1-3\alpha)k|$. This observation allows us to identify some strictly better vertices in the following. This is helpful when we wish to apply the second part of Lemma 8.6 on strictly better vertices.

Lemma 8.7. *Let $u, v \in V(G)$. Vertex v is strictly better than u if*

- *(Maximization:)* $\alpha \deg^{+c}(u) \leq \alpha \deg^{+c}(v) - |(1-3\alpha)k|$.
- *(Minimization:)* $\alpha \deg^{+c}(u) \geq \alpha \deg^{+c}(v) + |(1-3\alpha)k|$.

Proof. We give a proof for the maximization variant; the minimization variant follows analogously. By the definition of strictly better vertices, it suffices to show that $\mathrm{cont}(v,T) - \mathrm{cont}(u,T) \geq 0$ for each $T \subseteq V(G)$ of size at most k:

$$\begin{aligned}
&\mathrm{cont}(v,T) - \mathrm{cont}(u,T) \\
&= \alpha \deg^{+c}(v) + (1-3\alpha)|N(v) \cap T| - \alpha \deg^{+c}(u) - (1-3\alpha)|N(u) \cap T| \\
&= \alpha(\deg^{+c}(v) - \deg^{+c}(u)) + (1-3\alpha)(|N(v) \cap T| - |N(u) \cap T|) \\
&\geq |(1-3\alpha)k| + (1-3\alpha)(|N(v) \cap T| - |N(u) \cap T|) \geq |(1-3\alpha)k| + (1-3\alpha)k \geq 0.
\end{aligned}$$

This completes the proof. □

8.1.3 Reduction to Non-annotated Variant

The following two lemmas (for the maximization and the minimization variant, respectively) show that it is possible to remove annotations without blowing up the instance size. However, the instance size after removing annotations will depend on $\Delta_{\overline{T}} := \max_{v \in V(G) \setminus T} \deg(v)$ and $\Gamma := \max_{v \in V(G) \setminus T} \mathsf{counter}(v) + 1$. Note that the maximum degree $\Delta = \max_{v \in V(G)} \deg(V) \geq \Delta_{\overline{T}}$. We obtain an upper bound on Γ in terms of $k + \Delta$ in the next section.

Lemma 8.8. *Given an instance $\mathcal{I} := (G, T, \mathsf{counter}, k, t)$ of* Annotated Max α-FCGP *with $\alpha \in (0,1]$, we can compute an equivalent instance $\mathcal{I}'$ of* Max α-FCGP *of size $\mathcal{O}((\Delta_{\overline{T}} + \Gamma + \alpha^{-1}) \cdot |V(G)| + \alpha^{-1}k \cdot |T|))$ in polynomial time.*

Proof. We may assume that G has at least k vertices (otherwise no solution for Annotated Max α-FCGP exists and thus the empty graph and the same value of k form a no-instance for Max α-FCGP). We construct an equivalent instance $\mathcal{I}' := (G', k, t')$ of Max α-FCGP. The graph G' is obtained from G as follows:

1. Add $\mathsf{counter}(v) + \lfloor \alpha^{-1} \rfloor$ degree-one neighbors to every vertex $v \in V(G)$.

2. Additionally, add $\ell \coloneqq \Delta_{\overline{T}} + \Gamma + |\alpha^{-1} - 3| \cdot k + \lfloor \alpha^{-1} \rfloor$ degree-one neighbors to every vertex $v \in T$.

We denote by L_v the set of degree-one vertices added to vertex $v \in V(G)$ and we denote by $L \coloneqq \bigcup_{v \in V(G)} L_v$ the set of all newly added leaf vertices. To conclude the construction of $\mathcal{I}'$, we set $t' \coloneqq t + \alpha(\ell \cdot |T| + \lfloor \alpha^{-1} \rfloor \cdot k)$. Since G has at most $\Delta_{\overline{T}} \cdot |V(G)|$ edges and we add $\mathcal{O}((\Gamma + \alpha^{-1}) \cdot |V(G)| + (\Delta_{\overline{T}} + \Gamma + \alpha^{-1}k) \cdot |T|)$ edges, we see that G' has $\mathcal{O}((\Delta_{\overline{T}} + \Gamma + \alpha^{-1}) \cdot |V(G)| + \alpha^{-1}k \cdot |T|)$ edges.

Next, we prove the equivalence between $\mathcal{I}$ and $\mathcal{I}'$. For a solution S of $\mathcal{I}$, its value in G' is increased by $\alpha \cdot \lfloor \alpha^{-1} \rfloor$ for every vertex in S and, additionally, by $\alpha \cdot \ell$ for every vertex in T, amounting to $t + \alpha(\ell \cdot |T| + \lfloor \alpha^{-1} \rfloor \cdot k)$.

Conversely, consider a solution S' of $\mathcal{I}'$. First, we show that there is a solution containing all vertices of T and no leaf vertex of L using Lemma 8.6. Suppose that for some vertex $v \in V(G)$, one of its degree-one neighbors $v' \in L_v$ is in S' but not v itself. We then have $\mathrm{cont}(v', S' \setminus \{v\}) = \alpha$ and $\mathrm{cont}(v, S' \setminus \{v\}) \geq \alpha$, implying that $(S' \setminus \{v'\}) \cup \{v\}$ is also a solution by Lemma 8.6. Thus, in the following we can assume that $S' \cap L_v = \emptyset$ for every vertex $v \in V(G) \setminus S'$. If there is a vertex $v' \in S' \cap L_v$ for some $v \in S'$, then by the assumption that $|V(G)| \geq k$, the pigeonhole principle gives us a vertex $w \in V(G) \setminus S'$ with $S' \cap L_w = \emptyset$. Since $|L_w| \geq \mathsf{counter}(w) + \lfloor \alpha^{-1} \rfloor \geq \lfloor \alpha^{-1} \rfloor$, we have $\mathrm{cont}(w, S' \setminus \{v'\}) \geq \alpha \cdot \lfloor \alpha^{-1} \rfloor \geq \alpha(\alpha^{-1} - 1) = 1 - \alpha$. We thus have $\mathrm{cont}(v', S \setminus \{v'\}) = 1 - \alpha \leq \mathrm{cont}(w, S' \setminus \{v'\})$. Hence, $(S' \setminus \{v'\}) \cup \{w\}$ is a solution, again by Lemma 8.6.

Thus, we may assume that S' consists only of vertices from $V(G)$. Suppose that some vertex $v \in T$ is not in S'. For any vertex $v' \in S' \setminus T$, we have $\deg(v') \leq \deg_G(v) + \mathsf{counter}(v) + \lfloor \alpha^{-1} \rfloor \leq \Delta_{\overline{T}} + \Gamma + \lfloor \alpha^{-1} \rfloor$. So we have $\deg(v') \geq \Delta_{\overline{T}} + \Gamma + |\alpha^{-1}| \cdot k + \lfloor \alpha^{-1} \rfloor \geq \deg(v) + |\alpha^{-1} - 3| \cdot k$. Applying Lemma 8.7 with $\mathsf{counter}(v) = \mathsf{counter}(v') = 0$, we obtain that v is strictly better than v'. Now, it follows from Lemma 8.6 that $\mathcal{I}'$ has a solution S' such that $T \subseteq S' \subseteq V(G')$. Hence, S' is also a solution for $\mathcal{I}$. □

Lemma 8.9. *Given an instance $\mathcal{I} \coloneqq (G, T, \mathsf{counter}, k, t)$ of* Annotated Min α-FCGP *for $\alpha \in (0, 1]$, we can compute an equivalent instance $\mathcal{I}'$ of* Min α-FCGP *of size $\mathcal{O}(\alpha^{-2}(\Delta + \Gamma + k)^2 + \alpha^{-1}(\Delta + \Gamma + k) \cdot |V(G)|)$ in polynomial time.*

Proof. We may assume that G has at least k vertices (otherwise no solution for Annotated Min α-FCGP exists and the thus the empty graph and parameter k are a no-instance for Min α-FCGP). We construct an equivalent instance $\mathcal{I}' \coloneqq (G', k, t')$ of Min α-FCGP.

Let ℓ be the smallest integer greater than $\alpha^{-1}(\Delta + \Gamma + |(1 - 3\alpha)k|)$. Let G' be the graph obtained from G as follows: We add a clique C on $2\ell + 1$ vertices. For

every vertex $v \in V(G) \setminus T$, choose $\ell + \mathsf{counter}(v)$ vertices of C arbitrarily and add edges between v and the chosen vertices. To conclude the construction of $\mathcal{I}'$, we set $t' := t + \alpha\ell(k - |T|)$. Observe that we add $\mathcal{O}(\alpha^{-1}(\Delta + \Gamma + k))$ vertices and $\mathcal{O}(\alpha^{-2}(\Delta + \Gamma + k)^2 + \alpha^{-1}(\Delta + \Gamma + k) \cdot |V(G)|)$ edges.

Next, we show that $\mathcal{I}$ and $\mathcal{I}'$ are equivalent. For a solution S of $\mathcal{I}$ its value in G' is increased by $\alpha\ell$ for every vertex $v \in V(G) \setminus T$, amounting to $t + \alpha\ell(k - |T|)$. Thus, S is also a solution of I'.

Conversely, suppose that $\mathcal{I}'$ has a solution S'. We show that there is a solution that contains all vertices of T and no vertex of C. By construction, the following holds:

1. $\deg_{G'}(v) = \deg_G(v) \leq \Delta$ for any vertex $v \in T$.
2. $\deg_{G'}(v) = \deg_G(v) + \ell + \mathsf{counter}(v) \in [\ell, \Delta + \Gamma + \ell]$ for any vertex $v \in V(G) \setminus T$.
3. $\deg_{G'}(v) \geq 2\ell$ for any vertex $v \in C$.

Since $\ell \geq \alpha^{-1}(\Delta + \Gamma + |(1 - 3\alpha)k|)$, any vertex in T is strictly better than any vertex in $V(G) \setminus T$ and any vertex in $V(G) \setminus T$ is strictly better than any vertex in C by Lemma 8.7: To see the latter, consider $v_2 \in V(G) \setminus T$ and $v_3 \in C$. Then we have:

$$\begin{aligned}\alpha \deg_{G'}(v_3) - \alpha \deg_{G'}(v_2) &\geq \alpha 2\ell - \alpha(\Delta + \Gamma + \ell) \\ &= \alpha\ell - \alpha\Delta - \alpha\Gamma \\ &\geq \Delta + \Gamma + |(1 - 3\alpha)k| - \alpha\Delta - \alpha\Gamma \geq |(1 - 3\alpha)k|.\end{aligned}$$

Thus, by Lemma 8.6, $\mathcal{I}'$ admits a solution S' with $T \subseteq S' \subseteq V(G')$ and, hence, S' is a solution of value at least $t' - \alpha\ell(k - |T|) = t$ for $\mathcal{I}$. □

8.1.4 Dependence of the Kernel Sizes on α

To simplify notation, we will generally omit the polynomial factors in α^{-1} in the following sections. Note that when we remove the annotations using Lemma 8.8 or Lemma 8.9, a factor polynomial in α^{-1} appears in the size of the graph of the resulting α-FCGP instance. In our kernelization, we apply Lemma 8.8 or Lemma 8.9 once after obtaining an instance of Annotated α-FCGP in which the maximum degree Δ, the maximum counter Γ, and the graph G are all bounded by some (polynomial) function of the parameter in question.

In Sections 8.2 (maximum degree), 8.3 (closure), and 8.4 (degeneracy) the bound on Δ and the size of G will not depend on α^{-1}, while Γ has a term linearly dependent on α^{-1} (see Lemma 8.16). Thus, the kernel size will be proportional to α^{-1} for the

maximization variant (we remark that $\alpha^{-1} \leq 3$ for the degrading case) and α^{-4} for the minimization variant in the worst case. In Section 8.5 (vertex cover number and h-index), we make the dependence on α^{-1} explicit since many results have their own approach to obtain an upper bound on Δ and Γ.

8.2 Parameterization By Maximum Degree

8.2.1 Polynomial Kernels in Degrading Cases

Now, we present our framework to provide polynomial kernels of size $\Delta + k$. For this, it is essential to bound the largest counter of any vertex polynomial in $\Delta + k$. We do this by first adding vertices with a contribution far above t/k for maximization (and for below t/k for minimization) to the partial solution. Second, we remove vertices with contribution far below t/k for maximization (and far above t/k for minimization). We then show that this is sufficient to bound the counters.

To obtain a polynomial kernel for α-FCGP with respect to $\Delta + k$, we then show that it is sufficient to remove a vertex v if polynomial in $\Delta + k$ many vertices are better than v. To obtain kernels for the smaller parameters c-closure (Section 8.3) and degeneracy (Section 8.4) plus k it then remains to show that the maximum degree can be bounded in the parameter plus k.

Recall that in the degrading cases we have $\alpha \in (1/3, 1]$ for maximization and $\alpha \in [0, 1/3)$ for minimization. Furthermore, recall that for two vertices u and v, v is said to be better than u with respect to T if $\text{cont}(v, T) \geq \text{cont}(u, T)$ (vice versa for the minimization variant).

Recall that we defined $\Delta_{\overline{T}} \coloneqq \max_{v \in V(G) \setminus T} \deg(v)$ for the annotated version. To obtain the kernels in this section it is sufficient to use Δ instead of $\Delta_{\overline{T}}$ in Reduction Rule 8.3. However, in Section 8.5 it is sometimes important to use $\Delta_{\overline{T}}$ in Reduction Rule 8.3 to obtain the kernels.

Reduction Rule 8.3. *Let $\mathcal{I}$ be an instance of* Annotated Degrading α-FCGP. *If there are at least $(\Delta_{\overline{T}} - 1)(k-1) + 1$ vertices that are better than v with respect to T, then apply the Exclusion Rule (Reduction Rule 8.2) to v.*

Lemma 8.10. *Reduction Rule 8.3 is correct.*

Proof. Let $\mathcal{I}'$ be the reduced instance. Clearly, if $\mathcal{I}'$ has a solution S', then S' is also a solution for $\mathcal{I}$. Conversely, suppose that $\mathcal{I}$ has a solution S. If $v \notin S$, then S is also a solution for $\mathcal{I}'$. In the following, we assume that $v \in S$.

By the pigeonhole principle, there exists a vertex v' better than v such that $v \notin N[S \setminus T]$. We claim that $S' \coloneqq (S \setminus \{v\}) \cup \{v'\}$ is a solution for $\mathcal{I}'$. First, we consider maximization. By Lemma 8.6, it suffices to show that $\mathrm{cont}(v', S \setminus \{v\}) \geq \mathrm{cont}(v, S \setminus \{v\})$. Since $v' \notin N[S' \setminus T]$, we have $\mathrm{cont}(v', S \setminus \{v\}) = \mathrm{cont}(v', T) \geq \mathrm{cont}(v, T) \geq \mathrm{cont}(v, S \setminus \{v\})$. Here, the last inequality follows from the fact that the contribution is degrading.

Second, we consider minimization. By Lemma 8.6, it suffices to show that $\mathrm{cont}(v', S \setminus \{v\}) \leq \mathrm{cont}(v, S \setminus \{v\})$. Since $v' \notin N[S' \setminus T]$, we have $\mathrm{cont}(v', S \setminus \{v\}) = \mathrm{cont}(v', T) \leq \mathrm{cont}(v, T) \leq \mathrm{cont}(v, S \setminus \{v\})$. Here, the last inequality follows from the fact that the contribution is degrading. □

Next, we show that the exhaustive application of Reduction Rule 8.3 yields a polynomial kernel for Degrading α-FCGP.

Proposition 8.11. Degrading α-FCGP *has a kernel of size*

- $\mathcal{O}(\Delta^2 k)$ *for maximization and* $\alpha \in (1/3, 1]$, *and*
- $\mathcal{O}(\Delta k(\Delta + k))$ *for minimization and* $\alpha \in (0, 1/3)$.

Proof. Given an instance of Degrading α-FCGP, we transform it into an equivalent instance of Annotated Degrading α-FCGP and apply Reduction Rule 8.3 exhaustively. Observe that since Reduction Rule 8.3 is applied, we have $|V(G)| \leq \Delta k + 1$. Moreover, we have $T = \emptyset$ and $\Gamma \leq \Delta$ since each neighbor of a vertex can increase its counter by at most one. By Lemma 8.8 (maximization) or Lemma 8.9 (minimization), we obtain an equivalent instance of α-FCGP of size $\mathcal{O}(\Delta^2 k)$ (maximization) or $\mathcal{O}(\Delta k(\Delta + k))$ (minimization). □

Note that Proposition 8.11 does not cover the case $\alpha = 0$ for minimization, which is also called Sparsest k-Subgraph. The kernel for this case will be shown in Section 8.4: Proposition 8.34 provides a kernel of size $\mathcal{O}(d^2 k)$ for Sparsest k-Subgraph; since $d \leq \Delta$, this implies also a kernel of size $\mathcal{O}(\Delta^2 k)$.

Proposition 8.11 shows that given an instance of Degrading α-FCGP, we can find in polynomial time an equivalent instance of Degrading α-FCGP of size $\mathcal{O}(\Delta + k)^{\mathcal{O}(1)}$. In the following, in Proposition 8.18, we will show that an equivalent instance of Degrading α-FCGP that has size $(\Delta + k)^{\mathcal{O}(1)}$ can be constructed even if an instance of Annotated Degrading α-FCGP is given. Proposition 8.18 plays an important role in kernelizations in subsequent sections. Essentially, the task of kernelization for $k + c$ and $k + d$ boils down to bound the maximum degree Δ to apply Proposition 8.18.

As shown in the proof of Proposition 8.11, the number of vertices becomes polynomial in $k + \Delta$ by exhaustively applying Reduction Rule 8.3. Recall that in Section 8.1, we presented a polynomial-time procedure to remove annotations with an additional polynomial factor in $\Delta + \Gamma$ on the instance size, where Γ denotes the maximum counter. To prove Proposition 8.18, it remains to bound Γ for Annotated Degrading α-FCGP.

Bounding the largest counter Γ. Throughout the section, let $k' \coloneqq k - |T|$ and $t' \coloneqq t - \mathrm{val}(T)$. First, we identify some vertices which are contained in a solution, if one exists.

Definition 8.12. Let $\mathcal{I}$ be a yes-instance of Annotated Degrading α-FCGP. A vertex $v \in V(G) \setminus T$ is called *satisfactory* if

- (Maximization:) $\mathrm{cont}(v, T) \geq t'/k' + (3\alpha - 1)(k - 1)$ and $\alpha \in (1/3, 1]$.
- (Minimization:) $\mathrm{cont}(v, T) \leq t'/k' + (3\alpha - 1)(k - 1)$ and $\alpha \in (0, 1/3)$.

Reduction Rule 8.4. *Let $\mathcal{I}$ be an instance of* Annotated Degrading α-FCGP *with $\alpha > 0$ and let $v \in V(G) \setminus T$ be a satisfactory vertex. Apply the Inclusion Rule (Reduction Rule 8.1) on vertex v.*

Lemma 8.13. *Reduction Rule 8.4 is correct.*

Proof. Let $\mathcal{I}'$ be the reduced instance. Clearly, if $\mathcal{I}'$ has a solution S', then S' is also a solution for $\mathcal{I}$. Conversely, suppose that $\mathcal{I}$ has a solution S. The lemma clearly holds for $v \in S$. So we will assume that $v \notin S$. We start with an auxiliary claim:

Claim 11. There is an ordering $(v_1, \ldots, v_{k'})$ of the vertices of $S \setminus T$ with

- $\ell_1 \geq \ell_2 \geq \ldots \geq \ell_{k'}$ for maximization and $\alpha \in (1/3, 1]$, and
- $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_{k'}$ for minimization and $\alpha \in (0, 1/3)$,

where $S_i \coloneqq T \cup \{v_1, \ldots, v_{i-1}\}$ and $\ell_i \coloneqq \mathrm{cont}(v_i, S_i)$ for every $i \in [k']$.

Proof of Claim. Consider an ordering of $S \setminus T$, where the i-th vertex v_i is chosen in such a way that $\mathrm{cont}(v_i, S_i)$ is maximized (minimized). Then, since the contribution is degrading, we conclude that $\mathrm{cont}(v_i, S_i) \geq \mathrm{cont}(v_{i+1}, S_i) \geq \mathrm{cont}(v_{i+1}, S_{i+1})$ for maximization, and $\mathrm{cont}(v_i, S_i) \leq \mathrm{cont}(v_{i+1}, S_i) \leq \mathrm{cont}(v_{i+1}, S_{i+1})$ for minimization, respectively, for every $i \in [k' - 1]$. ■

Note that $S_{k'} = S \setminus \{v_{k'}\}$. Now, consider the vertex set $S' \coloneqq S_{k'} \cup \{v\}$. We show that $\mathrm{val}(S') \geq t$ for maximization and that $\mathrm{val}(S') \leq t$ for minimization.

By Lemma 8.2, we have $\mathrm{val}(S') = \mathrm{val}(S) - \ell_{k'} + \mathrm{cont}(v, S_{k'})$. The definition of contribution yields that

$$\begin{aligned}\mathrm{cont}(v, S_{k'}) &= \alpha \deg^{+c}(v) + (1-3\alpha)|N(v) \cap S_{k'}| \\ &= \mathrm{cont}(v, T) + (1-3\alpha)|N(v) \cap (S_{k'} \setminus T)|.\end{aligned}$$

Since $|S_{k'} \setminus T| = k' - 1$ and $\mathrm{cont}(v, T) \geq t'/k' + (3\alpha - 1)(k-1)$ for maximization and $\mathrm{cont}(v, T) \leq t'/k' + (3\alpha - 1)(k-1)$ for minimization, we have

$$\begin{aligned}\mathrm{cont}(v, S_{k'}) \geq [t'/k' + (3\alpha-1)(k-1)] + (1-3\alpha)(k'-1) \geq t'/k' \\ \text{for maximization and } \alpha \in (1/3, 1], \text{ and} \\ \mathrm{cont}(v, S_{k'}) \leq [t'/k' + (3\alpha-1)(k-1)] + (1-3\alpha)(k'-1) \leq t'/k' \\ \text{for minimization and } \alpha \in (0, 1/3).\end{aligned}$$

For maximization, if $\ell_{k'} \leq t'/k'$, then we have $\ell_{k'} \leq \mathrm{cont}(v, S_{k'})$ and thus $\mathrm{val}(S') \geq \mathrm{val}(S) \geq t$. Analogously, for minimization, if $\ell_{k'} \geq t'/k'$, then we have $\ell_{k'} \geq \mathrm{cont}(v, S_{k'})$ and thus $\mathrm{val}(S') \leq \mathrm{val}(S) \leq t$. Thus, in the following, we assume that $\ell_{k'} > t'/k'$ for maximization and that $\ell_{k'} < t'/k'$ for minimization. We obtain that

$$\begin{aligned}&\mathrm{val}(S_{k'}) = \mathrm{val}(T) + \sum_{i=1}^{k'-1} \ell_i. \text{ Thus,} \\ &\mathrm{val}(S_{k'}) \geq \mathrm{val}(T) + (k'-1)\ell_{k'} \geq \mathrm{val}(T) + \frac{(k'-1)t'}{k'} \text{ for maximization, and} \\ &\mathrm{val}(S_{k'}) \leq \mathrm{val}(T) + (k'-1)\ell_{k'} \leq \mathrm{val}(T) + \frac{(k'-1)t'}{k'} \text{ for minimization.}\end{aligned}$$

Hence, for maximization $\mathrm{val}(S') = \mathrm{val}(S_{k'}) + \mathrm{cont}(v, S_{k'}) \geq [\mathrm{val}(T) + (k'-1)t'/k'] + t'/k' = t$ and for minimization $\mathrm{val}(S') = \mathrm{val}(S_{k'}) + \mathrm{cont}(v, S_{k'}) \leq [\mathrm{val}(T) + (k'-1)t'/k'] + t'/k' = t$. □

We henceforth assume that Reduction Rule 8.4 is exhaustively applied on every satisfactory vertex. Next, we identify some vertices which are not contained in any solution.

Definition 8.14. Let $\mathcal{I}$ be a yes-instance of Annotated Degrading α-FCGP. A vertex $v \in V(G) \setminus T$ is called *needless* if

- (Maximization:) $\mathrm{cont}(v, T) < t'/k' - (3\alpha-1)(k-1)^2$ for maximization and $\alpha \in (1/3, 1]$.

- (Minimization:) $\text{cont}(v,T) > t'/k' - (3\alpha-1)(k-1)^2$ for minimization and $\alpha \in (0,1/3)$.

Reduction Rule 8.5. *Let $\mathcal{I}$ be an instance of* Annotated Degrading α-FCGP *with $\alpha > 0$ and let $v \in V(G) \setminus T$ be a needless vertex. Apply the Exclusion Rule (Reduction Rule 8.2) on vertex v.*

Lemma 8.15. *Reduction Rule 8.5 is correct.*

Proof. Let $\mathcal{I}'$ be the reduced instance. Clearly, if $\mathcal{I}'$ has a solution S', then S' is also a solution for $\mathcal{I}$. Conversely, suppose that $\mathcal{I}$ has a solution S. The lemma clearly holds for $v \notin S$. So we will assume that $v \in S$. Since Reduction Rule 8.4 is exhaustively applied to each satisfactory vertex v', we obtain that $\text{cont}(v',T) < t'/k' + (3\alpha-1)(k-1)$ for maximization and $\text{cont}(v',T) > t'/k' + (3\alpha-1)(k-1)$ for minimization, for each vertex $v' \in V(G) \setminus T$. Together with Lemma 8.2 we thus obtain for maximization that

$$\begin{aligned}\text{val}(S) &\leq \text{val}(T) + \text{cont}(v,T) + (k'-1)[t'/k' + (3\alpha-1)(k-1)] \\ &< \text{val}(T) + [t'/k' - (3\alpha-1)(k-1)^2] + (k'-1)[t'/k' + (3\alpha-1)(k-1)] \\ &= \text{val}(T) + t' - (3\alpha-1)(k-1)(k-k') \leq t \text{ since } \alpha \in (1/3, 1].\end{aligned}$$

Similarly, for minimization, we obtain that

$$\begin{aligned}\text{val}(S) &\geq \text{val}(T) + \text{cont}(v,T) + (k'-1)[t'/k' + (3\alpha-1)(k-1)] \\ &> \text{val}(T) + [t'/k' - (3\alpha-1)(k-1)^2] + (k'-1)[t'/k' + (3\alpha-1)(k-1)] \\ &= \text{val}(T) + t' + (1-3\alpha)(k-1)(k-k') \geq t \text{ since } \alpha \in (0, 1/3).\end{aligned}$$

For maximization, this is a contradiction to $\text{val}(S) \geq t$, and for minimization this is a contradiction to $\text{val}(S) \leq t$. Thus, S cannot contain the needless vertex v. □

We henceforth assume that Reduction Rule 8.5 is applied on every needless vertex. The following reduction rule decreases the counter of each vertex in $V(G) \setminus T$. After this rule is exhaustively applied, we may assume that $\mathsf{counter}(v) = 0$ for at least one vertex $v \in V(G) \setminus T$. Recall that we already have $\mathsf{counter}(v) = 0$ for every vertex in T.

Reduction Rule 8.6. *If $\mathsf{counter}(v) > 0$ for every vertex $v \in V(G) \setminus T$, then decrease $\mathsf{counter}(v)$ by 1 for every vertex $v \in V(G) \setminus T$ and decrease t by $\alpha k'$.*

Next, we show that after the exhaustive application of Reduction Rule 8.6 the counter of each vertex is bounded polynomially in terms of Δ and k.

Lemma 8.16. *Let $\mathcal{I}$ be a reduced yes-instance of* Annotated Degrading α-FCGP *with $\alpha > 0$. We have* $\mathsf{counter}(v) \in \mathcal{O}(\Delta + \alpha^{-1}k^2)$ *for every vertex $v \in V(G) \setminus T$.*

Proof. First, observe that there exists at least one vertex $u \in V(G) \setminus T$ with $\mathsf{counter}(u) = 0$, since otherwise Reduction Rule 8.6 is still applicable. Since Reduction Rule 8.5 is applied to each needless vertex, we conclude that every vertex in $V(G) \setminus T$ has contribution at least $t'/k' - (3\alpha - 1)(k-1)^2$ for maximization. Furthermore, since Reduction Rule 8.4 is applied to each satisfactory vertex, we conclude that every vertex $v \in V(G) \setminus T$ has contribution at least $t'/k' + (3\alpha - 1)(k-1)$ for minimization. In particular, we have

$$\begin{aligned} &\mathrm{cont}(u,T) \geq t'/k' - (3\alpha-1)(k-1)^2 \text{ for maximization, and} \\ &\mathrm{cont}(u,T) \geq t'/k' + (3\alpha-1)(k-1) \text{ for minimization.} \end{aligned}$$

Since also $\mathrm{cont}(u,T) = \alpha \cdot \deg(u) + (1-3\alpha)|N(u) \cap T|$ we obtain that

$$t'/k' \leq \alpha \cdot \deg(u) + (1-3\alpha)|N(u) \cap T| + (3\alpha-1)(k-1)^2 \text{ for maximization, and} \tag{8.1}$$

$$t'/k' \geq \alpha \cdot \deg(u) + (1-3\alpha)[(k-1) + |N(u) \cap T|] \text{ for minimization.} \tag{8.2}$$

Moreover, since Reduction Rule 8.1 is applied to each satisfactory vertex, we conclude that every vertex $v \in V(G) \setminus T$ has contribution at most $t'/k' + (3\alpha - 1)(k-1)$ for maximization. Furthermore, since Reduction Rule 8.2 is applied to each needless vertex, we conclude that every vertex in $V(G) \setminus T$ has contribution at most $t'/k' - (3\alpha-1)(k-1)^2$ for minimization. This implies that in particular

$$\alpha \cdot \mathsf{counter}(v) \leq t'/k' + (3\alpha-1)(k-1) \text{ for maximization, and.} \tag{8.3}$$

$$\alpha \cdot \mathsf{counter}(v) \leq t'/k' - (3\alpha-1)(k-1)^2 \text{ for minimization.} \tag{8.4}$$

For maximization and $\alpha \in (1/3, 1]$ it then follows from Equations (8.1) and (8.3) that

$$\mathsf{counter}(v) \leq \deg(u) + \frac{3\alpha-1}{\alpha}[k(k-1) - |N(v) \cap T|] \in \mathcal{O}(\Delta + \alpha^{-1}k^2).$$

For minimization and $\alpha \in (0, 1/3)$ it then follows from Equations (8.2) and (8.4) that

$$\mathsf{counter}(v) \leq \deg(u) + \frac{1-3\alpha}{\alpha}[k(k-1) + |N(v) \cap T|] \in \mathcal{O}(\Delta + \alpha^{-1}k^2).$$

This concludes the proof. □

Putting everything together. For the kernels, we first transform the instance into an equivalent instance of Annotated Degrading α-FCGP. Second, we apply our reduction rules. For the third step, we use the following proposition to reduce back to the unannotated version.

Proposition 8.17. *Let $\alpha > 0$.*

- *Given an instance $(G, T, \mathsf{counter}, k, t)$ of* Annotated Degrading Max α-FCGP*, we can compute in polynomial time an equivalent* Degrading Max α-FCGP *instance of size $\mathcal{O}(|V(G)|^2 + \alpha^{-1}|V(G)|k^2) \subseteq \mathcal{O}(\alpha^{-1}|V(G)|^3)$.*
- *Given an instance $(G, T, \mathsf{counter}, k, t)$ of* Annotated Degrading Min α-FCGP*, we can compute in polynomial time an equivalent instance of* Degrading Min α-FCGP *of size $\mathcal{O}(\alpha^{-2}(|V(G)| + \alpha^{-1}k^2)^2) \subseteq \mathcal{O}(\alpha^{-4}|V(G)|^4)$.*

Proof. Using Reduction Rules 8.1, 8.2, and 8.6 exhaustively yields, by Lemma 8.16, an instance where $\mathsf{counter}(v) \in \mathcal{O}(\Delta + \alpha^{-1}k^2) \subseteq \mathcal{O}(|V(G)| + \alpha^{-1}k^2)$. For maximization, by Lemma 8.8 we get an equivalent instance of Degrading Max α-FCGP of size

$$\mathcal{O}(|V(G)|^2 + |V(G)|k^2 + k^2) \subseteq \mathcal{O}(\alpha^{-1}|V(G)|^3), \text{ and}$$

for minimization, by Lemma 8.9 we get an equivalent instance of Degrading Min α-FCGP of size

$$\begin{aligned}\mathcal{O}(\alpha^{-2}(|V(G)| + \alpha^{-1}k^2)^2 + \alpha^{-1}(|V(G)| + \alpha^{-1}k^2)|V(G)|) \\ \subseteq \mathcal{O}(\alpha^{-2}(|V(G)| + \alpha^{-1}k^2)^2) \\ \subseteq \mathcal{O}(\alpha^{-4}|V(G)|^4).\end{aligned}$$

Hence, the statement follows. □

Proposition 8.18. *Given an instance $(G, T, \mathsf{counter}, k, t)$ of* Annotated Degrading α-FCGP *with $\alpha > 0$, we can compute in polynomial time an equivalent* Degrading α-FCGP *instance of size $(\Delta + k)^{\mathcal{O}(1)}$.*

Proof. First, we bound $\mathsf{counter}(v)$ by $(\Delta + k)^{\mathcal{O}(1)}$ due to Lemma 8.16. Second, we obtain a kernel of size $(\Delta + k)^{\mathcal{O}(1)}$ due to Proposition 8.11. Finally, we transform the resulting instance in an equivalent instance of Degrading α-FCGP of size $(\Delta + k)^{\mathcal{O}(1)}$ due to Lemma 8.8 (maximization) or Lemma 8.9 (minimization). □

8.2.2 No Polynomial Kernels in Non-Degrading Cases

Note that if $\alpha = 0$, then Max α-FCGP corresponds to Densest k-Subgraph. It is already known that Densest k-Subgraph does not admit a polynomial kernel for $\Delta + k$ [150]. We strengthen and generalize this result: First, we observe that Densest k-Subgraph does not admit a polynomial kernel for k, even when $\Delta = 3$. Second, we extend this negative result to Non-Degrading Max α-FCGP and Non-Degrading Min α-FCGP when Δ is a constant.

Theorem 8.19. *Unless* coNP $\subseteq$ NP/poly*,*

1. *For each* $\alpha \in [0, 1/3)$*,* Non-Degrading Max α-FCGP *on subcubic graphs does not admit a polynomial kernel for* k*, and*

2. *For each* $\alpha \in (1/3, 1]$*,* Non-Degrading Min α-FCGP *on graphs with constant maximum degree does not admit a polynomial kernel for* k*.*

Proof. We present a polynomial-parameter transformation from Clique parameterized by the size of the largest connected component (this parameterization does not admit a polynomial kernel which can be seen by an or-composition of disjoint Clique instances with the same clique size) to Non-Degrading Max α-FCGP (and Non-Degrading Min α-FCGP) parameterized by the solution size k. The reduction is based on a reduction of Feige and Seltser [77] that shows NP-hardness of Densest k-Subgraph on subcubic graphs with a subsequent reduction to Non-Degrading Max α-FCGP and Non-Degrading Min α-FCGP, respectively.

The reduction from Clique to Densest k-Subgraph works as follows. Let (G, ℓ) be an instance of Clique. Without loss of generality, we may assume that every connected component of G has exactly $\hat{n}$ vertices and more than $\binom{\ell}{2}$ edges. For each connected component H of G perform the following construction. Let $\{v_1, \ldots, v_{\hat{n}}\}$ denote the vertex set of H. For each vertex v_i of H create a cycle $C_i := (v_i^1, \ldots, v_i^{\hat{n}})$ of length $\hat{n}$. This cycle is called the *vertex cycle of* v_i. Then, add a path on $\hat{n}^2 + 1$ edges between v_r^q and v_q^r if v_r and v_q are adjacent in H. In the following, this path is called *connector path* of v_q^r and v_r^q. To each newly added vertex on this connector path, attach a degree-one vertex. Furthermore, attach a degree-one vertex to each vertex on a vertex cycle which has no connector path. We call these newly added degree-one vertices *pendant*. Observe that every non-pendant vertex has degree three. Let G' denote the constructed graph.

Claim 12. The graph G contains a clique of size ℓ if and only if G' contains a $k' := \ell\hat{n} + \binom{\ell}{2}\hat{n}^2$-vertex subgraph with at least $\delta' := \ell\hat{n} + \binom{\ell}{2}(\hat{n}^2 + 1)$ edges.

Proof of Claim. Suppose that G contains a clique K of size ℓ. Then, consider the induced subgraph that contains the ℓ vertex cycles of the clique vertices and all connector paths between them. This subgraph has the claimed number of vertices and edges since the number of cycles is ℓ and since each pair of cycles is connected via a path because K is a clique.

Conversely, assume that G' contains a k'-vertex subgraph $G'[S]$ with at least δ' edges. First, observe that we may assume that S contains no pendant vertices: By construction and the fact that each connected component of G' has more than $\binom{\ell}{2}$ edges, each non-pendant vertex of G' is connected to more than k' non-pendant vertices. Hence, every pendant vertex of S can be replaced by a non-pendant neighbor $u \notin S$ of some non-pendant vertex $v \in S$ without decreasing the number of edges in $G'[S]$.

After this observation, the correctness proof can be carried out in the same manner as in the proof of Feige and Seltser [77]. We sketch the details for sake of completeness. First, observe that every degree-one vertex u of $G'[S]$ may reach at most one degree-3 vertex v of $G'[S]$ via some path that contains only vertices that have degree 2 in $G'[S]$. We say that the degree-one vertex u is *associated* with the degree-3 vertex v. We call a degree-3 vertex *good* if it is not associated with any degree-one vertex. A simple calculation shows that $G'[S]$ must contain at least $2\binom{\ell}{2}$ good degree-3 vertices to achieve the claimed number of edges (refer to Feige and Seltser [77] for further details). If a degree-3 vertex is good, then $G'[S]$ contains its connector path completely. By the choice of k' and the path length, $G'[S]$ may contain at most $\binom{\ell}{2}$ connector paths completely and thus both endpoints of a connector path that is contained completely in $G'[S]$ are good. Now, let $\mathcal{P}$ denote the set of connector paths that are completely contained in $G'[S]$ and let $\mathcal{C}$ denote the collection of vertex cycles that contain a good vertex or, equivalently, that contain an endpoint of a connector path in $\mathcal{P}$.

We show that we may assume that every vertex cycle of $\mathcal{C}$ is completely contained in $G'[S]$. Assume, towards a contradiction, that $G'[S]$ does not contain all vertex cycles of $\mathcal{C}$ completely and that $G'[S]$ is a k'-vertex and δ'-edge subgraph of G with the largest possible number of vertices of $\mathcal{C}$. Choose some vertex cycle C_v of $\mathcal{C}$ that is not contained completely in $G'[S]$. By definition of $\mathcal{C}$, this vertex cycle contains a good vertex u. Since C_v is not contained completely in $G'[S]$, some vertex y of C_v has degree one in $G'[S]$ and is connected to u via a path of vertices of C_v. Since u is good, there must be some vertex w on this path which has degree 3 in $G'[S]$ but which is not good. Thus, $G'[S]$ contains some but not all vertices of the connector path starting at w. Let x be the degree-one vertex on this connector path that is contained in $G'[S]$ and reachable from w via the connector path vertices. Removing x

from $G'[S]$ and adding the missing cycle neighbor of y gives a graph $\tilde{G}$ with the same number of edges and vertices but with more cycle vertices, a contradiction to the choice of $G'[S]$.

Consequently, we may assume that $G'[S]$ contains all cycles of $\mathcal{C}$ completely. By the choice of k' we have that $|\mathcal{C}| = \ell$. Thus, $G'[S]$ contains ℓ vertex cycles with $\binom{\ell}{2}$ connector paths between them. The vertex set of G corresponding to the vertex cycles is thus a clique of size ℓ. ■

The proof of Claim 12 also implies the correctness of the following observation.

Claim 13. If G' contains an $\ell\hat{n}+\binom{\ell}{2}\hat{n}^2$-vertex subgraph with at least $\ell\hat{n}+\binom{\ell}{2}(\hat{n}^2+1)$ edges, then it contains one such subgraph that has no pendant vertices. In other words, such a size $\ell\hat{n}+\binom{\ell}{2}\hat{n}^2$-vertex subgraph contains only vertices of degree 3.

No polynomial kernel for Non-Degrading Max α-FCGP Observe that the size parameter k' of the Densest-k-Subgraph instance depends only on $\hat{n}$ and that the maximum degree of G' is 3. Moreover, Claim 12 shows the correctness of the reduction. We thus have shown the theorem statement for Densest-k-Subgraph, the special case of Non-Degrading Max α-FCGP with $\alpha = 0$. We now provide a polynomial-time transformation from the Densest-k-Subgraph instance (G', k', δ') to an instance (G', k', t) of Non-Degrading Max α-FCGP with arbitrary $\alpha \in [0, 1/3)$ by setting $t \coloneqq (1-\alpha)\delta' + \alpha(3k' - 2\delta')$. Since G' and k' are not changed by this transformation, it only remains to show its correctness.

Assume that (G', k', δ') is a yes-instance, and let S be a size-k' vertex set such that $G'[S]$ has at least δ' edges. By Claim 13, we may assume that S contains only vertices that have degree 3 in G'. Thus, in G', we have $\mathrm{val}(S) = (1-\alpha)m(S) + \alpha m(S, V(G') \setminus S) = (1-\alpha)m(S) + \alpha(3k' - 2m(S)) \geq (1-\alpha)\delta' + \alpha(3k' - 2\delta')$ since $m(S) \geq \delta'$ and $(1-\alpha) > 2\alpha$.

Conversely, let S be a size-k' set with $\mathrm{val}(S) \geq t = (1-\alpha)\delta' + \alpha(3k' - 2\delta')$ in G'. First, observe that we may assume that S does not contain pendant vertices: Each such vertex has degree one in G' and thus degree at most one in $G'[S]$. Such a vertex exists since each connected component of G has at least $\binom{\ell}{2}$ edges and hence each connected component of G' has at least k' vertices. Consequently, it can be replaced by some non-pendant neighbor of a vertex in S. Consequently, $\mathrm{val}(S) = (1-\alpha)m(S) + \alpha(3k' - 2m(S))$. As in the proof of the forward direction, it follows that $m(S) \geq \delta'$.

No polynomial kernel for Non-Degrading Min α-FCGP Finally, we provide a polynomial-parameter transformation from the Densest-k-Subgraph in-

stance (G', k', δ') to an equivalent instance (G'', k', t) of Non-Degrading Min α-FCGP. In the construction, we distinguish whether $\alpha \in (1/3, 1)$ or $\alpha = 1$. In both cases, we add a gadget to each pendant vertex of G' such that each size-k' set S with minimal value $\mathrm{val}(S)$ in G'' contains only vertices that have degree at most 3 in G'', which will be exactly the vertices of G'. Hence, for both cases it remains to show that any solution with minimal value $\mathrm{val}(S)$ contains only non-pendant vertices of G'. Afterwards, the proof of the equivalence of (G, ℓ) and (G'', k', t) is the same as for Non-Degrading Max α-FCGP.

First, we show the statement for $\alpha \in (1/3, 1)$. Construct G'' from G' by adding for each pendant vertex u in G' a set K_u on $\lceil 15/(1-\alpha) \rceil$ vertices and making $K_u \cup \{u\}$ a clique. Set $t \coloneqq (1-\alpha)\delta' + \alpha(3k' - 2\delta')$. Assume towards a contradiction that some set S of minimal value contains a vertex u with $\deg(u) > 3$ in G''. By construction u is part of some clique K_y where y is some pendant vertex of G'. Let $Y \coloneqq K_y \cap S$ and let $r \coloneqq |Y|$. We show that for $S' \coloneqq (S \setminus Y) \cup Z$ for any vertex set $Z \subseteq V(G') \setminus S$ consisting of r non-pendant vertices we have $\mathrm{val}(S') < \mathrm{val}(S)$, contradicting the minimality of $\mathrm{val}(S)$. Observe that such a vertex set Z exists since each connected component of G' has at least $\binom{\ell}{2}$ edges and thus each connected component of G'' has at least k'' non-pendant vertices. Since $|K_y| = \lceil 15/(1-\alpha) \rceil$, we obtain that $\mathrm{val}(S \setminus Y) \leq \mathrm{val}(S) - (1-\alpha)\binom{r}{2} - \alpha r \lceil 15/(1-\alpha) \rceil$. Furthermore, observe that adding the vertices in Z increases the objective value by at most $3r$ since each non-pendant vertex has degree 3 in G''. Also, note that removing Y from S may result in a new outgoing edge of S which is incident with the neighbor of y in $V(G')$. Hence, we obtain that

$$\mathrm{val}((S \setminus Y) \cup Z) \leq \mathrm{val}(S) - (1-\alpha)\binom{r}{2} - \alpha r \lceil 15/(1-\alpha) \rceil + 3r + \alpha.$$

Now, if $r \leq \lceil 15/(1-\alpha) \rceil - 4$, then $-\alpha r \lceil 15/(1-\alpha) \rceil + 3r + \alpha < 0$ and thus $\mathrm{val}((S \setminus Y) \cup Z) < \mathrm{val}(S)$, a contradiction to the minimality of S. Otherwise, if $r \geq \lceil 15/(1-\alpha) \rceil - 3$, then $(1-\alpha)\binom{r}{2} > (1-\alpha)\lceil 11/(1-\alpha) \rceil \lceil 10/(1-\alpha) \rceil / 2 > 55/(1-\alpha) > 3 \cdot \lceil 15/(1-\alpha) \rceil + \alpha$ and thus $\mathrm{val}((S \setminus Y) \cup Z) < \mathrm{val}(S)$, a contradiction to the minimality of S. Hence, S contains only vertices of G'.

Second, we show the statement for $\alpha = 1$ which corresponds to Min $(k, n-k)$-Cut. Construct G'' from G' by adding for each degree-one vertex u in G' a graph H to G'' and making an arbitrary vertex $h \in V(H)$ adjacent to u. The graph H is a $(|V(G')|, q, \rho)$-edge expander, that is, H has $|V(G')|$ vertices, is q-regular for some constant q and every vertex set $T \subseteq V(H)$ of size at most $|V(G')|/2$ fulfills $m(T, V(H) \setminus T) \geq \rho q |T|$. We choose the $(|V(G')|, q, \rho)$-edge expander H in

such a way that $\rho q \geq 4$. Note that such a graph H exists and can be constructed in polynomial-time [9, 114]. Finally, we set $t \coloneqq (1-\alpha)\delta' + \alpha(3k' - 2\delta')$. Assume towards a contradiction that some set S of minimal value contains a vertex u with $\deg(u) > 3$ in G''. By construction, u is part of some copy of H which was attached to a pendant (degree-one) vertex y of G'. Let $Y \coloneqq H \cap S$ and let $r \coloneqq |Y|$.

Since H is a $(|V(G')|, q, \rho)$-edge expander and since $k' < |V(G'')|/2$ we obtain that $m(Y, V(G'') \setminus S) \geq 4r$. Since each vertex in G' has degree at most 3, we thus obtain for any set $Z \subseteq V(G')$ of size r that $\mathrm{val}((S \setminus Y) \cup Z) \leq \mathrm{val}(S \setminus Y) - 4r + 3r < \mathrm{val}(S)$, a contradiction to the minimality of S. Hence, S contains only vertices of G'. □

8.3 Parameterization by c-Closure

Now, we show that the maximum degree Δ can be bounded by $k^{\mathcal{O}(c)}$ in the degrading variant. This then gives us a kernel of size $k^{\mathcal{O}(c)}$. To prove this result we rely on a polynomial bound on the Ramsey bound in c-closed graphs. Then, we show that these kernels cannot be improved under standard assumptions. Finally, we provide W[1]-hardness for the non-degrading variant.

8.3.1 A Tight $k^{\mathcal{O}(c)}$-size Kernel for the Degrading Case

We develop a tight kernel of size $k^{\mathcal{O}(c)}$ for the degrading case.

A $k^{\mathcal{O}(c)}$-size kernel for the degrading case. To this end, we apply a series of reduction rules to obtain an upper bound of $k^{\mathcal{O}(c)}$ on the maximum degree. Then, the kernel of size $k^{\mathcal{O}(c)}$ follows from Proposition 8.18. In order to upper-bound the maximum degree, we rely on a polynomial Ramsey bound for c-closed graphs [139].

Lemma 8.20 ([139, Lemma 3.1])**.** *Any c-closed graph G on at least $R_c(q, b) \coloneqq (c-1) \cdot \binom{b-1}{2} + (q-1)(b-1) + 1$ vertices contains a clique of size q or an independent set of size b. Moreover, a clique of size q or an independent set of size b can be found in polynomial time.*

Using a similar approach as Reduction Rule 8.3 (but exploiting the c-closure instead of the maximum degree) yields the following.

Reduction Rule 8.7. *Let $\mathcal{I}$ be an instance of* Annotated Degrading α-FCGP*. Let $v \in V(G)$ be some vertex and let $X_v \subseteq N(v)$ be the set of vertices better than v. If $|X_v| > (c-1)k$, then apply the Exclusion Rule (Reduction Rule 8.2) to v.*

Lemma 8.21. *Reduction Rule 8.7 is correct.*

Proof. We provide a proof for the maximization version; the minimization version follows analogously. Let S be a solution. Assume that $v \in S$ (we are done otherwise). We show that there is a vertex $v' \neq v$ such that $S' \coloneqq (S \setminus \{v\}) \cup \{v'\}$ constitutes a solution. By Lemma 8.6, it suffices to show that $\mathrm{cont}(v', S \setminus \{v\}) \geq \mathrm{cont}(v, S \setminus \{v\})$. Let $S'_v \coloneqq S \setminus N[v]$. Each vertex in S'_v is, by definition, non-adjacent to v, and hence it shares at most $c-1$ neighbors with v. This implies $|X_v \setminus N(S'_v)| \geq |X_v| - (c-1) \cdot |S'_v| > 0$ as $X_v \subseteq N(v)$. Thus, there exists a vertex $v' \in X_v \setminus N(S'_v)$. Note in particular that $N(v') \cap S'_v = \emptyset$. Then, we have $N(v') \cap (S \setminus \{v\}) \subseteq S \cap N(v)$ and thus $|N(v') \cap (S \setminus \{v\})| \leq |N(v) \cap (S \setminus \{v\})|$. Moreover, we have $\alpha \deg^{+c}(v') \geq \alpha \deg^{+c}(v)$ (recall that v' is better than v). Since $\alpha \in (1/3, 1]$, it follows that

$$\begin{aligned}\mathrm{cont}(v', S \setminus \{v\}) &= \alpha \deg^{+c}(v') + (1-3\alpha)|N(v') \cap (S \setminus \{v\})| \\ &\geq \alpha \deg^{+c}(v) + (1-3\alpha)|N(v) \cap (S \setminus \{v\})| = \mathrm{cont}(v, S \setminus \{v\}).\end{aligned}$$

□

Note that if there is a clique of size $(c-1)k+1$, then Reduction Rule 8.7 applies to one of the vertices with the smallest contribution. Thus, applying Reduction Rule 8.7 exhaustively removes all cliques of size $(c-1)k+1$. By Lemma 8.20, if there is a vertex v with sufficiently large neighborhood, then we find a large independent set in $N(v)$. We can then identify a vertex for which we can apply Reduction Rule 8.5.

Lemma 8.22. *Suppose that $\Delta \geq R_c((c-1)k+1, (k+1)k^{c-2})$. Then, we can find in polynomial time a set X of $i \in [c-1]$ vertices and an independent set I with the following properties:*

1. *The set $I \subseteq \bigcap_{x \in X} N(x)$ is an independent set of size at least $(k+1)k^{c-i}$.*
2. *For every vertex $u \in V(G) \setminus X$, it holds that $|N(u) \cap I| \leq (k+1)k^{c-i-1}$.*

Proof. Let v be a vertex such that $\deg(v) \geq R_c((c-1)k+1, (k+1)k^{c-2})$. Since there is no clique of size $(c-1)k+1$, there is, by Lemma 8.20, an independent set I_v of size $(k+1)k^{c-2}$ in $N(v)$, which can be found in polynomial time. Let X be an inclusion-wise maximal set of i vertices including v such that $|N^{\cap}(X) \cap I_v| > (k+1)k^{c-i}$. Such a set can be found by the following polynomial-time algorithm: We start with $X \coloneqq \{v\}$ and $i \coloneqq 1$. We will maintain the invariant that $|X| = i$. If there exists a vertex $v' \in V(G) \setminus X$ with $|N(v') \cap N^{\cap}(X) \cap I_v| > (k+1)k^{c-i-1}$, then we add v' to X and increase i by 1. We keep doing so until there remains no such vertex v'.

We show that this algorithm terminates for $i = |X| \leq c - 1$. Assume to the contrary that the algorithm continues for $i = c - 1$. We then have that $|N(v') \cap N^{\cap}(X) \cap I_v| > (k+1)k^{c-i-1} = k + 1 \geq 2$ for some vertex $v' \in V(G) \setminus X$. Since I_v is an independent set, the set $N(v') \cap N^{\cap}(X) \cap I_v$ contains two non-adjacent vertices. Note, however, that these two vertices have at least $|X \cup \{v\}| = c$ common neighbors, contradicting the c-closure of G.

Finally, we show that the set X found by this algorithm and $I \coloneqq N^{\cap}(X) \cap I_v$ satisfy the three properties of the lemma. We have $|N^{\cap}(x) \cap I_v| = |N(v') \cap N^{\cap}(X \setminus \{v\}) \cap I_v| > (k+1)k^{c-(i-1)-1} = (k+1)k^{c-i}$, where v' is the last vertex added to X. Moreover, since X is inclusion-wise maximal, we have $|N(u) \cap I| = |N(u) \cap N^{\cap}(X) \cap I_v| \leq (k+1)k^{c-i-1}$ for every vertex $u \in V(G) \setminus X$. □

Reduction Rule 8.8. *Let $\mathcal{I}$ be an instance of* Annotated Degrading α-FCGP. *Let X, I be as specified in Lemma 8.22 and let $v \in I$ be a vertex such that every other vertex in I is better than v. If $k \geq 2$, then apply the Exclusion Rule (Reduction Rule 8.2) to v.*

Lemma 8.23. *Reduction Rule 8.8 is correct.*

Proof. Again, we show the proof for the maximization variant; the minimization variant follows analogously. For the sake of contradiction, assume that every solution S contains v. By Lemma 8.22, every vertex $u \in V(G) \setminus X$ has at most $(k+1)k^{c-i-1}$ neighbors in I. Moreover, since I is an independent set, we have $|I \cap N[v']| = 1$ for every vertex $v' \in I$ (including v). For $S' \coloneqq S \setminus X$, we have

$$\begin{aligned} |I \setminus N[S']| &\geq |I| - \sum_{u \in S' \setminus \{v\}} |I \cap N[u]| - |I \cap N[v]| \\ &\geq (k+1)k^{c-i} - (k-1)(k+1)k^{c-i-1} - 1 = k^{c-i} + k^{c-i-1} - 1 > 0. \end{aligned}$$

Let v' be an arbitrary vertex in $I \setminus N[S']$. We show that $\mathrm{cont}(v', S \setminus \{v\}) \geq \mathrm{cont}(v, S \setminus \{v\})$. By Lemma 8.6, this will imply that $(S \setminus \{v\}) \cup \{v'\}$ is a solution not containing v. Since v and v' are both adjacent to all vertices of X and $\alpha \in (1/3, 1]$, we have $|N(v) \cap (S \setminus \{v\})| > |X \cap (S \setminus \{v\})|$. We thus have

$$\begin{aligned} \mathrm{cont}(v', S \setminus \{v\}) &= \alpha \deg^{+c}(v') + (1 - 3\alpha)|X \cap (S \setminus \{v\})| \\ &\geq \alpha \deg^{+c}(v) + (1 - 3\alpha)|X \cap (S \setminus \{v\})| \\ &\geq \alpha \deg^{+c}(v) + (1 - 3\alpha)|N(v) \cap (S \setminus \{v\})| = \mathrm{cont}(v, S \setminus \{v\}). \end{aligned}$$

Here, the first inequality follows from the fact that v' is better than v. □

By applying these reduction rules exhaustively, we obtain an instance with $\Delta \leq R_c((c-1)k+1,(k+1)k^{c-2}) \in k^{\mathcal{O}(c)}$. Proposition 8.18 then leads to the following:

Proposition 8.24. DEGRADING α-FCGP *has a kernel of size* $k^{\mathcal{O}(c)}$.

Matching lower bounds. Next, we show that the kernel provided in Proposition 8.24 cannot be improved under standard assumptions.

Proposition 8.25. DEGRADING MAX α-FCGP *has no kernel of size* $\mathcal{O}(k^{c-3-\epsilon})$ *unless* coNP $\subseteq$ NP/poly.

We first prove the following technical lemma. This lemma is useful to obtain an upper bound (maximization) or a lower bound (minimization) for $\mathrm{val}(S)$ if the set of vertices (denoted as C^* in the lemma) in the *instance choice gadget* is fixed and we only have to choose vertices D^* in the *instance gadgets* (the union of all the instance vertices is D in the lemma).

Lemma 8.26. *Let* (G,k,t) *be an instance of* DEGRADING α-FCGP *with a solution* S *fulfilling* $C^* \subseteq S \subseteq C^* \cup D$ *where* D *is a set of vertices of the same degree. Let* $D^* \coloneqq S \cap D$. *Then:*

1. *For maximization,* $\mathrm{val}(S)$ *is maximal if* $m(D^*) + m(D^*, C^*)$ *is minimal.*
2. *For minimization,* $\mathrm{val}(S)$ *is minimal if* $m(D^*) + m(D^*, C^*)$ *is minimal.*

Proof. Let $z \coloneqq \deg(p)$ for each $d \in D$, let $D^* \coloneqq \{d_1, \ldots, d_\ell\}$, and let $D_i \coloneqq \{d_j \mid j < i\}$. By Lemma 8.2 we obtain that

$$\begin{aligned}
\mathrm{val}(S) &= \mathrm{val}(C^*) + \sum_{i=1}^{\ell} \mathrm{cont}(d_i, C^* \cup D_i) \\
&= \mathrm{val}(C^*) + \alpha z \ell + (1-3\alpha) \sum_{i=1}^{\ell} |N(d_i) \cap C^*| + |N(d_i) \cap D_i| \\
&= \mathrm{val}(C^*) + \alpha z \ell + (1-3\alpha)\big(m(D^*, C^*) + m(D^*)\big)
\end{aligned}$$

Note that $\mathrm{val}(C^*) + \alpha z \ell$ is a constant. Thus, for maximization in the degrading case we obtain that $\mathrm{val}(S)$ is maximized if $m(D^*, C^*) + m(D^*)$ is minimized since $(1 - 3\alpha) < 0$. Furthermore, for minimization in the degrading case we obtain that $\mathrm{val}(S)$ is minimized if $m(D^*, C^*) + m(D^*)$ is minimized since $(1-3\alpha) > 0$. □

Proof of Proposition 8.25. We provide a weak q-composition from Independent Set on 2-closed graphs to Degrading Max α-FCGP in $(q+3)$-closed graphs. Here, we assume that $k > 3q + \lceil \frac{\alpha}{3\alpha-1} \rceil$.

Independent Set
Input: A graph $G = (V, E)$ and an integer k.
Question: Is there an independent set of size exactly k?

Let $[t]^q$ be the set of q-dimensional vectors whose entries are in $[t]$. For a vector $x \in [t]^q$ we denote by x_i the i-th entry of x. Assume that $q \geq 2$ is a constant and that we are given exactly $[t]^q$ instances $\mathcal{I}_x \coloneqq (G_x, k)$ of Independent Set on 2-closed graphs. Let $V_x \coloneqq V(G_x)$ for each $x \in [t]^q$ and let $D \coloneqq \bigcup_{x \in [t]^q} V_x$. We construct an equivalent instance (H, k', t') of Degrading Max α-FCGP as follows.

Construction: First, for each $x \in [t]^q$ we add the graph G_x to H. In other words, we added the *instance gadgets* to H. We then add a clique C (the *instance choice gadget*) consisting of tq vertices to H. The vertices of C are denoted by w^i_j with $i \in [q]$ and $j \in [t]$. Furthermore, for each $x \in [t]^q$ and $v \in V_x$, we add the edge $vw^i_{x_i}$ for each $i \in [q]$. Fix an integer $\ell \geq t^q \cdot n + (\alpha k + k + 1) \cdot \alpha^{-1}$. We add leaf vertices so that $\deg(w^i_j) = \ell + \lceil (3\alpha - 1)tq \cdot \alpha^{-1} \rceil$ for each vertex w^i_j and $\deg(v) = \ell$ for each vertex $v \in D$. We denote the union of all these added leaf vertices by L. Finally, we set $k' \coloneqq k + tq - q$ and

$$t' \coloneqq (1-\alpha)\frac{(tq-q)(tq-q-1)}{2} + \alpha\left[k\ell + (tq-q)\left(\ell + \left\lceil \frac{(3\alpha-1)tq}{\alpha} \right\rceil\right) - (tq-q)(tq-q-1)\right].$$

Closure: We show that H is $(q+3)$-closed. Since leaf vertices have degree one and C is a clique, we only have to consider non-adjacent vertex pairs where one vertex u is in D and the other vertex v is in $C \cup D$. Without loss of generality we assume that $u \in V_x$ for some $x \in [t]^q$. Recall that $N(u) \subseteq V_x \cup C \cup L$ and u has exactly q neighbors in C. First, consider the case $v \in C$. By construction we obtain from $uv \notin E$ that $u'v \notin E$ for each $u' \in V_x$. Thus, $|N(u) \cap N(v)| = q$. Second consider the case $v \in D$. If $v \notin V_x$, then $|N(u) \cap N(v)| \leq q - 1$. Otherwise, if $v \in V_x$, we obtain $|N(u) \cap N(v)| \leq q + 2$ from the fact that $\mathcal{I}_x$ is 2-closed.

Correctness: In the following, we prove that there exists an independent set of size exactly k for some instance $\mathcal{I}_x$ with $x \in [t]^q$ if and only if there exists a vertex set S of size exactly k' in H such that $\mathrm{val}(S) \geq t'$.

Suppose that instance $\mathcal{I}_x$ has an independent set I of size exactly k for some $x \in [t]^q$. By $C^* \coloneqq C \setminus \{w^i_{x_i} \mid i \in [q]\}$ we denote the non-neighbors of V_x in C. Note that $|C^*| = tq - q$. We show that $S \coloneqq I \cup C^*$ is a solution of (H, k', t'). Clearly, $|S| = k + tq - q = k'$. Since no vertex of I is connected with any vertex in C^*, I is an independent set, and since C^* is a clique of size $tq - q$, we conclude that $m_H(S) = (tq-q)(tq-q-1)/2$. Furthermore, since each vertex in I has degree ℓ, and since each vertex in C^* has degree $\ell + \lceil (3\alpha - 1)tq \cdot \alpha^{-1} \rceil$, we conclude that $m_H(S, V(H) \setminus S) = k\ell + (tq - q)(\ell + \lceil (3\alpha - 1)tq \cdot \alpha^{-1} \rceil) - (tq - q)(tq - q - 1)$. Thus, $\mathrm{val}(S) = t'$ and hence (H, k', t') is a yes-instance of Degrading Max α-FCGP.

Conversely, suppose that (H, k', t') has a solution $S \subseteq V(H)$ of size exactly k' with $\mathrm{val}(S) \geq t'$. First, we show that we can assume that $S \cap L = \emptyset$. Assume that there exists a vertex $v \in S \cap L$ and let $w \in (C \cup D) \setminus S$. We show that for $S' \coloneqq S \setminus \{v\} \cup \{w\}$ we have $\mathrm{val}(S') > \mathrm{val}(S)$. Observe that $\mathrm{val}(S \setminus \{v\}) \geq \mathrm{val}(S) - 1$. Furthermore, note that adding w may result into at most k new inner edges and hence the value decreases by at most k. Simultaneously, since $\deg(w) \geq \ell$, at least $\ell - k$ new outer edges emerge such that the value increases by at least $\alpha(\ell - k)$. Since $\ell > (\alpha k + k + 1) \cdot \alpha^{-1}$, we obtain $\mathrm{val}(S') > \mathrm{val}(S)$.

Thus, in the following we can assume that $S \cap L = \emptyset$. Let $C^* \coloneqq C \cap S$, $|C^*| = z$, and $D^* \coloneqq S \setminus C^* \subseteq D$. In the following, we show that $z = tq - q$ and that there exists an $x \in [t]^q$ such that $N(V_x) \cap C^* = \emptyset$. For this, we consider the cases that $z < tq - q$ and that $z > tq - q$. In both cases we verify that $\mathrm{val}(S) < t$ for each solution with exactly z vertices in C.

Case 1: $z \leq tq - q - 1$. In other words, $z = tq - q - p$ for some $p \in [tq - q]$. Recall that $\deg(v) = \ell$ for each vertex $v \in D$. Thus, by Lemma 8.26, $\mathrm{val}(S)$ is maximized if $m_H(D^*) + m_H(C^*, D^*)$ is minimized. Since $z = |C^*| < tq - q$, it is possible that no vertex of D^* is adjacent to any vertex in C^*. Thus, $\mathrm{val}(S)$ is maximized if D^* is an independent set and if $E_H(C^*, D^*) = \emptyset$. Hence,

$$\begin{aligned}\mathrm{val}(S) \leq (1-\alpha)&\frac{(tq-q-p)(tq-q-p-1)}{2} \\ &+ \alpha\left[(k+p)\ell + (tq-q-p)\left(\ell + \left\lceil\frac{(3\alpha-1)tq}{\alpha}\right\rceil\right)\right] \\ &- \alpha(tq-q-p)(tq-q-p-1) \eqqcolon f(p).\end{aligned}$$

Now, we obtain that the derivative f' of f is

$$f'(p) = (1-3\alpha)p + \frac{3\alpha-1}{2}(2tq-2q-1) - \alpha\Big\lceil\frac{(3\alpha-1)tq}{\alpha}\Big\rceil.$$

Since $(1-3\alpha) < 0$ for $\alpha \in (1/3, 1]$, we obtain for $p \leq tq - q$ that

$$\begin{aligned} f'(p) &\leq \frac{3\alpha-1}{2}(2tq-2q-1) - \alpha\Big\lceil\frac{(3\alpha-1)tq}{\alpha}\Big\rceil \\ &\leq \frac{3\alpha-1}{2}(2tq-2q-1) - (3\alpha-1)tq = (1-3\alpha)(q+1/2) < 0. \end{aligned}$$

Since $\alpha \in (1/3, 1]$ we conclude that $f(p)$ is a concave quadratic function. And since $f'(p) < 0$ for each $p \leq tq - q$, we thus conclude that $f(0) > f(p)$ for each $p \in [tq - q]$, a contradiction to the assumption that $f(p) = \mathrm{val}(S) \geq t'$.

Case 2: $z \geq tq - q + 1$**.** Let $z = tq - q + p$ for some $p \in [q]$. By the pigeonhole principle there exist at least p indices $i \in [q]$ such that $w_j^i \in S$ for each $j \in [t]$. Recall that by construction, each vertex $v \in D$ has exactly one neighbor in the set $\{w_j^i, j \in [t]\}$. Since $|D^*| = k - p$ we conclude that $m_H(C^*, D^*) \geq (k-p)p$. Recall that $\deg(v) = \ell$ for each vertex $v \in D$. Thus, by Lemma 8.26, $\mathrm{val}(S)$ is maximal if $m_H(D^*) + m_H(C^*, D^*)$ is minimal. Hence, D^* is an independent set and $m_H(C^*, D^*) = (k-p)p$. Thus,

$$\begin{aligned} \mathrm{val}(S) \leq{}& (1-\alpha)\left[(k-p)p + \frac{(tq-q+p)(tq-q+p-1)}{2}\right] + \alpha(k-p)(\ell-p) \\ &+ \alpha p\left[\ell + \Big\lceil\frac{(3\alpha-1)tq}{\alpha}\Big\rceil - tq + q - k + 1\right] \\ &+ \alpha(t-1)q\left[\ell + \Big\lceil\frac{(3\alpha-1)tq}{\alpha}\Big\rceil - tq + q - p + 1\right] =: f(p). \end{aligned}$$

Now, we obtain that the derivative f' of f is

$$f'(p) = (3\alpha-1)p + \Big(1-3\alpha\Big)\Big(k + tq - q - \frac{1}{2}\Big) + \alpha\Big\lceil\frac{(3\alpha-1)tq}{\alpha}\Big\rceil.$$

Since $\alpha \in (1/3, 1]$ and since $p \in [q]$, we obtain that

$$\begin{aligned} f'(p) &\leq (3\alpha-1)q + (1-3\alpha)(k + tq - q - 1/2) + \alpha\Big(\frac{(3\alpha-1)tq}{\alpha} + 1\Big) \\ &\leq (3\alpha-1)\Big(2q + 1/2 + \frac{\alpha}{3\alpha-1} - k\Big). \end{aligned}$$

Since $k > 3q + \lceil \frac{\alpha}{3\alpha-1} \rceil$ and since $\alpha \in (1/3, 1]$, we obtain that $f'(p) < 0$ for $p \leq q$. Furthermore, since $\alpha \in (1/3, 1]$ we see that $f(p)$ is a convex quadratic function. Thus, we conclude that $f(0) > f(p)$ for each $p \in [q]$, a contradiction to the assumption that $f(p) = \mathrm{val}(S) \geq t'$.

Hence, $|C^*| = tq - q$ and thus $|D^*| = k$. According to Lemma 8.26, $\mathrm{val}(S)$ is maximal if $m_H(D^*) + m_H(C^*, D^*)$ is minimal. Observe that if D^* is an independent set and if $E_H(C^*, D^*) = \emptyset$, then $\mathrm{val}(S) = t'$. Otherwise, if $E_H(C^*, D^*) \neq \emptyset$ or if D^* is no independent set, then $\mathrm{val}(S) < t'$. Thus, $E_H(C^*, D^*) = \emptyset$ and D^* is an independent set. Now, if there exist two vertices $u, v \in D^*$ such that $u \in V_x$ and $v \in V_y$ with $x \neq y$ for $x, y \in [t]^q$, then $(N(u) \cup N(v)) \cap C \geq q + 1$. Since $|C^*| = tq - q$ this implies that $E_H(C^*, D^*) \neq \emptyset$, a contradiction. Hence, $D^* \subseteq V_x$ for some $x \in [t]^q$. Since D^* is an independent set, we conclude that the instance x contains an independent set of size at least k.

Hence, we have a weak-q-composition from Independent Set to Degrading Max α-FCGP in $(q + 3)$-closed graphs. Now, the statement of the proposition follows by Lemma 2.14. □

Proposition 8.27. *For each $\alpha \in (0, 1/3)$,* Min α-FCGP *does not admit a kernel of size $\mathcal{O}(k^{c-3-\epsilon})$ unless* coNP $\subseteq$ NP/poly.

Proof. The proof is similar to the proof of Proposition 8.25: The main difference is that we now cannot add leaf vertices to ensure that all vertices in the instance gadgets (and also all vertices in the instance choice gadget) have the same degree since then an optimal solution would simply pick these leaf vertices. Instead, we add large cliques to ensure that all vertices in the instance gadgets (and also all vertices in the instance choice gadget) have the same degree. We provide a weak q-composition from Independent Set on 2-closed graphs to Min α-FCGP in $(q + 3)$-closed graphs. Here, we assume that $k > 3q + \lceil \frac{\alpha}{1-3\alpha} \rceil$.

Independent Set
Input: A graph $G = (V, E)$ and an integer k.
Question: Is there an independent set of size exactly k?

Let $[t]^q$ be the set of q-dimensional vectors whose entries are in $[t]$. For a vector $x \in [t]^q$ we denote by x_i the i-th entry of x. Assume that $q \geq 2$ is a constant and that we are given exactly $[t]^q$ instances $\mathcal{I}_x := (G_x, k)$ of Independent Set on 2-closed graphs. Let $V_x := V(G_x)$ for each $x \in [t]^q$ and let $D := \bigcup_{x \in [t]^q} V_x$. We construct an equivalent instance (H, k', t') of Min α-FCGP as follows.

Construction: First, for each $x \in [t]^q$ we add the graph G_x to H. In other words, we added the *instance gadgets* to H. We then add a clique C (the *instance choice gadget*) consisting of tq vertices to H. The vertices of C are denoted by w^i_j with $i \in [q]$ and $j \in [t]$. Furthermore, for each $x \in [t]^q$ and $v \in V_x$, we add the edge $vw^i_{x_i}$ for each $i \in [q]$. Fix an integer $\ell \geq t^q \cdot n + \lceil (1-3\alpha)tq \cdot \alpha^{-1} \rceil$. We add leaf vertices so that $\deg(w^i_j) = \ell - \lceil (1-3\alpha)tq \cdot \alpha^{-1} \rceil$ for each vertex w^i_j and $\deg(v) = \ell$ for each vertex $v \in D$. Next, for each added leaf vertex v, we add a clique C_v of size $\lceil 3t' \cdot \alpha^{-1} \rceil$ to H. By L we denote the set of newly added leaf vertices and vertices in the cliques C_v for the new leaf vertices. Finally, we set $k' \coloneqq k + tq - q$ and

$$t' \coloneqq (1-\alpha)\frac{(tq-q)(tq-q-1)}{2} + \alpha\left[k\ell + (tq-q)\left(\ell - \left\lceil \frac{(1-3\alpha)tq}{\alpha} \right\rceil\right) - (tq-q)(tq-q-1)\right].$$

Closure: We show that H is $(q+3)$-closed. Note that $H[L]$ is a cluster graph, that is, a graph in which each connected component is a clique, and thus has closure one. Furthermore, observe that each vertex in L has at most one neighbor in $V(H) \setminus L$. From now on the argumentation is completely analogous to the argumentation in Proposition 8.25. We thus omit it.

Correctness: In the following, we prove that there exists an independent set of size exactly k for some instance $\mathcal{I}_x$ with $x \in [t]^q$ if and only if there exists a vertex set S of size exactly k' in H such that $\mathrm{val}(S) \leq t'$.

Suppose that instance $\mathcal{I}_x$ has an independent set I of size exactly k for some $x \in [t]^q$. By $C^* \coloneqq C \setminus \{w^i_{x_i} \mid i \in [q]\}$ we denote the non-neighbors of V_x in C. Note that $|C^*| = (t-1)q$. We show that $S \coloneqq I \cup C^*$ is a solution of (H, k', t'). Clearly, $|S| = k + tq - q = k'$. Since no vertex of I is connected with any vertex in C^*, I is an independent set, and since C^* is a clique of size $(t-1)q$, we conclude that $m_H(S) = (tq-q)(tq-q-1)/2$. Furthermore, since each vertex in I has degree ℓ, and since each vertex in C^* has degree $\ell - \lceil (1-3\alpha)tq \cdot \alpha^{-1} \rceil$, we conclude that $m_H(S, V(H) \setminus S) = k\ell + (tq-q)(\ell - \lceil (1-3\alpha)tq \cdot \alpha^{-1} \rceil) - (tq-q)(tq-q-1)$. Thus, $\mathrm{val}(S) = t'$ and hence (H, k', t') is a yes-instance of Min α-FCGP.

Conversely, suppose that (H, k', t') has a solution $S \subseteq V(H)$ of size exactly k' with $\mathrm{val}(S) \leq t'$. First, we show that we can assume that $S \cap L = \emptyset$. Assume that there exists a vertex $v \in S \cap L$. Since $\deg(v) = \lceil 3t' \cdot \alpha^{-1} \rceil$ we conclude that $m_H(\{v\}, S \setminus \{v\}) \geq 2t'\alpha$ and since $\mathrm{val}(S) \geq \alpha \cdot m_H(\{v\}, S \setminus \{v\})$ we obtain that $\mathrm{val}(S) > t'$, a contradiction.

Thus, in the following we can assume that $S \cap L = \emptyset$. Let $C^* \coloneqq C \cap S$, $|C^*| = z$, and $D^* \coloneqq S \setminus C^* \subseteq D$. Next, we show that $z = tq - q$ and that there exists an $x \in [t]^q$ such that $N(V_x) \cap C^* = \emptyset$. To show that, we consider the cases that $z < tq - q$ and that $z > tq - q$. In both cases we verify that $\mathrm{val}(S) > t$ for each solution with exactly z vertices in C.

Case 1: $z \leq tq - q - 1$**.** In other words, $z = (t-1)q - p$ for some $p \in [tq - q]$. Recall that $\deg(v) = \ell$ for each vertex $v \in D$. Thus, by Lemma 8.26, $\mathrm{val}(S)$ is minimized if $m_H(D^*) + m_H(C^*, D^*)$ is minimized. Since $z = |C^*| < tq - q$, it is possible that no vertex of D^* is adjacent to any vertex in C^*. Thus, $\mathrm{val}(S)$ is minimized if D^* is an independent set and if $E_H(C^*, D^*) = \emptyset$. Hence,

$$\begin{aligned}\mathrm{val}(S)) \geq (1-\alpha)&\frac{(tq-q-p)(tq-q-p-1)}{2}\\ &+\alpha\left[(k+p)\ell + (tq-q-p)\left(\ell - \left\lceil\frac{(1-3\alpha)tq}{\alpha}\right\rceil\right)\right]\\ &-\alpha(tq-q-p)(tq-q-p-1) \eqqcolon f(p).\end{aligned}$$

Now, we obtain that the derivative f' of f is

$$f'(p) = (1-3\alpha)p + \frac{3\alpha-1}{2}\Big(2tq - 2q - 1\Big) + \alpha\left\lceil\frac{(1-3\alpha)tq}{\alpha}\right\rceil.$$

Since $\alpha \in (0, 1/3)$ and since $p \leq tq - q$, we obtain that

$$f'(p) \geq \frac{3\alpha-1}{2}\Big(2tq - 2q - 1\Big) + \alpha\frac{(1-3\alpha)tq}{\alpha} = (1-3\alpha)(q+1/2) > 0.$$

Since $\alpha \in [0, 1/3]$ we conclude that $f(p)$ is a convex quadratic function. And since $f'(p) > 0$ for each $p \leq q$, we thus conclude that $f(0) < f(p)$ for each $p \in [q]$, a contradiction to the assumption that $f(p) = \mathrm{val}(S) \leq t'$.

Case 2: $z \geq tq - q + 1$**.** Let $z = tq - q + p$ for some $p \in [q]$. By the pigeonhole principle there exist at least p indices $i \in [q]$ such that $w^i_j \in S$ for each $j \in [t]$. Recall that by construction, each vertex $v \in D$ has exactly one neighbor in the set $\{w^i_j, j \in [t]\}$. Since $|D^*| = k - p$ we conclude that $m_H(C^*, D^*) \geq (k-p)p$. Recall that $\deg(v) = \ell$ for each vertex $v \in D$. Thus, by Lemma 8.26, $\mathrm{val}(S)$ is

minimal if $m_H(D^*) + m_H(C^*, D^*)$ is minimal. Hence, D^* is an independent set and $m_H(C^*, D^*) = (k-p)p$. Thus,

$$\begin{aligned}\mathrm{val}(S) \geq (1-\alpha)&\left[(k-p)p + \frac{(tq-q+p)(tq-q+p-1)}{2}\right] + \alpha(k-p)(\ell-p)\\ &+ \alpha p\left[\ell - \left\lceil\frac{(1-3\alpha)tq}{\alpha}\right\rceil - tq + q - k + 1\right]\\ &+ \alpha(tq-q)\left[\ell - \left\lceil\frac{(1-3\alpha)tq}{\alpha}\right\rceil - tq + q - p + 1\right] =: f(p).\end{aligned}$$

Now, we obtain that the derivative f' of f is

$$f'(p) = (3\alpha-1)p + \Big(1-3\alpha\Big)\Big(k + tq - q - \frac{1}{2}\Big) - \alpha\left\lceil\frac{(1-3\alpha)tq}{\alpha}\right\rceil.$$

From $p \leq q$ and $3\alpha - 1 < 0$ for $\alpha \in (0, 1/3)$, we obtain that

$$\begin{aligned}f'(p) &\geq (3\alpha-1)q + (1-3\alpha)\Big(k + tq - q - 1/2\Big) - \alpha\Big(\frac{(1-3\alpha)tq}{\alpha} + 1\Big)\\ &= (1-3\alpha)\Big(k - 2q - 1/2 - \frac{\alpha}{1-3\alpha}\Big).\end{aligned}$$

Since $k > 3q + \lceil\frac{\alpha}{1-3\alpha}\rceil$ and since $\alpha \in (0, 1/3)$, we obtain that $f'(p) > 0$ for $p \leq q$. Furthermore, since $\alpha \in (0, 1/3)$ we see that $f(p)$ is a concave quadratic function. Thus, we conclude that $f(0) < f(p)$ for each $p \in [q]$, a contradiction to the assumption that $f(p) = \mathrm{val}(S) \leq t'$.

Hence, $|C^*| = tq - q$ and thus $|D^*| = k$. According to Lemma 8.26, $\mathrm{val}(S)$ is minimal if $m_H(D^*) + m_H(C^*, D^*)$ is minimal. Observe that if D^* is an independent set and if $E_H(C^*, D^*) = \emptyset$, then $\mathrm{val}(S) = t'$. Otherwise, if $E_H(C^*, D^*) \neq \emptyset$ or if D^* is no independent set, then $\mathrm{val}(S) > t'$. Thus, $E_H(C^*, D^*) = \emptyset$ and D^* is an independent set. Now, if there exist two vertices $u, v \in D^*$ such that $u \in V_x$ and $v \in V_y$ with $x \neq y$ for $x, y \in [t]^q$, then $(N(u) \cup N(v)) \cap C \geq q + 1$. Since $|C^*| = tq - q$ this implies that $E_H(C^*, D^*) \neq \emptyset$, a contradiction. Hence, $D^* \subseteq V_x$ for some $x \in [t]^q$. Since D^* is an independent set, we conclude that the instance x contains an independent set of size at least k.

Hence, we have a weak-q-composition from Independent Set to Min α-FCGP in $(q+3)$-closed graphs. Now, the proposition follows by Lemma 2.14. □

Recall that Min α-FCGP for $\alpha = 0$ is equivalent to Sparest k-Subgraph which admits a kernel of size $\mathcal{O}(c^2k^3)$ [137].

Now, Propositions 8.24, 8.25, and 8.27 imply the following.

Theorem 8.28. Degrading α-FCGP *admits a kernel of size* $k^{\mathcal{O}(c)}$. *For* $\alpha > 0$, Degrading α-FCGP *does not admit a kernel of size* $k^{o(c)}$ *unless* coNP $\subseteq$ NP/poly.

8.3.2 Hardness for the Non-Degrading Case

In contrast to the degrading case, we show that the non-degrading case is intractable.

Theorem 8.29. Non-Degrading Max α-FCGP *remains W[1]-hard with respect to the solution size* k *even on 2-closed and 2-degenerate graphs.*

Proof. We reduce from the W[1]-hard Densest-k-Subgraph problem in 2-closed and 2-degenerate graphs [198]. Recall that Densest k-Subgraph is the special case of Max α-FCGP with $\alpha = 0$. Hence, it remains to show the theorem for $\alpha \in (0, 1/3)$. Let (G, k, t) be an instance of Densest k-Subgraph. We construct an equivalent instance (G', k', t') of Max α-FCGP as follows: Initially, graph G' consists of a copy of G. Let Z denote the set of all vertices which are a copy of a vertex in G. We add exactly $n(G) - \deg_G(v)$ many leaf-vertices to each vertex $v \in Z$. We denote these vertices by I_v. By $I \coloneqq \bigcup_{v \in V(G)} I_v$ we denote the set of all these leaf-vertices. Finally, we set $k' \coloneqq k$ and $t' \coloneqq \alpha(kn(G) - 2t) + (1 - \alpha)t$.

By construction, each vertex in I has degree 1. Since G is 2-closed and 2-degenerate, we conclude that also G' is 2-closed and 2-degenerate.

Let $S \subseteq V(G)$ be such that $|S| = k$ and that $m(G[S]) \geq t$. Since

- each vertex $v \in V(G') \cap Z$ has degree $n(G)$,
- $N(x) \cap N(y) \subseteq Z$ for each two vertices $x, y \in Z$, and
- $m(G[S]) \geq t$,

we conclude that exactly $x \geq t$ edges have both endpoints in S and that exactly $kn(G) - 2x \leq kn(G) - 2t$ edges have exactly one endpoint in S. Thus, $\mathrm{val}(S) \geq \alpha(kn(G) - 2t) + (1 - \alpha)t = t'$ since $\alpha \in (0, 1/3)$.

Conversely, suppose that $S' \subseteq V(G')$ is set of exactly k vertices with $\mathrm{val}(S') \geq t'$. By construction, each vertex in I has degree 1. Hence, $\mathrm{cont}(v) \leq \max(\alpha, 1 - \alpha) < 1$ for each vertex $v \in I$. Furthermore, observe that for each vertex $z \in Z$ we have $\mathrm{cont}(w) \geq \alpha n(G) > 1$ since α is a constant. Thus, $\mathrm{cont}(z) > \mathrm{cont}(v)$ for each vertex $z \in Z$ and each $v \in I$. Hence, we can assume that $S' \subseteq Z$. Let x be the

number of edges with both endpoints in S'. Since each vertex in S' has degree $n(G)$, we conclude that exactly $kn(G) - 2x$ edges have exactly one endpoint in S'. Hence, $\mathrm{val}(S') = \alpha(kn(G) - 2x) + (1-\alpha)x$. Since $\mathrm{val}(S') \geq t'$ and $\alpha \in (0, 1/3)$, we conclude that $x \geq t$. Hence, $G[S']$ contains at least t edges. □

Theorem 8.30. Non-Degrading Min α-FCGP *remains W[1]-hard with respect to the solution size k even on 2-closed graphs.*

Proof. We reduce from the W[1]-hard Densest-k-Subgraph problem in 2-closed graphs [198]. Let (G, k, t) be an instance of Densest k-Subgraph. We construct an equivalent instance (G', k', t') of Min α-FCGP as follows: Initially, graph G' consists of a copy of G. Let Z denote the set of all vertices which are a copy of a vertex in G. For each vertex $v \in Z$ we add a set W_v of exactly $n(G) - \deg_G(v)$ many vertices to G' such that each vertex of W_v is adjacent with vertex v. By $W \coloneqq \bigcup_{v \in V(G)} W_v$ we denote the set of all these vertices. Next, for each vertex $w \in W$ we add a clique C_w of size $4t'$ to G' such that each vertex of C_w is adjacent with w. By $C \coloneqq \bigcup_{w \in W} C_w$ we denote the set of all these vertices. Finally, we set $k' \coloneqq k$ and $t' \coloneqq \alpha(kn(G) - 2t) + (1-\alpha)t$.

Observe that $G'[W \cup C]$ is a cluster graph, that is, a disjoint union of cliques. Furthermore, observe that each vertex in $W \cup C$ has at most one neighbor in Z. Since $G[Z]$ is 2-closed, we conclude that G' is 2-closed.

Let $S \subseteq V(G)$ be such that $|S| = k$ and that $m(G[S]) \geq t$. Since

- each vertex $v \in V(G') \cap Z$ has degree $n(G)$,
- $N(x) \cap N(y) \subseteq Z$ for each two vertices $x, y \in Z$, and
- $m(G[S]) \geq t$,

we conclude that exactly $x \geq t$ edges have both endpoints in S and that exactly $kn(G) - 2x \leq kn(G) - 2t$ edges have exactly one endpoint in S. Thus, $\mathrm{val}(S) \leq \alpha(kn(G) - 2t) + (1-\alpha)t = t'$ since $\alpha \in (1/3, 1]$.

Conversely, suppose that $S' \subseteq V(G')$ is set of exactly k vertices with $\mathrm{val}(S') \leq t'$. By construction, each vertex in $W \cup C$ has degree at least $2t'$. If S' contains a vertex v of $W \cup C$, then v has at least $3t'$ neighbors which are not in S' since $t' > k$. Thus, $\mathrm{val}(S') > \alpha 3t' > t'$, a contradiction to the assumption $\mathrm{val}(S') \leq t'$. Hence, $S' \cap (W \cup C) = \emptyset$. Let x be the number of edges with both endpoints in S'. Since each vertex in S' has degree $n(G)$, we conclude that exactly $kn(G) - 2x$ edges have exactly one endpoint in S'. Hence, $\mathrm{val}(S') = \alpha(kn(G) - 2x) + (1-\alpha)x$. Since $\mathrm{val}(S') \leq t$ and $\alpha \in (1/3, 1]$, we conclude that $x \geq t$. Hence, $G[S']$ contains at least t edges. □

8.4 Parameterization by Degeneracy

We show that in the minimization variant we obtain an FPT-algorithm for each α. For minimization in the degrading variant we even obtain polynomial kernels for $d+k$. In contrast, for maximization in the degrading variant we provide a tight kernel of size $k^{\mathcal{O}(d)}$. To prove this result we again rely on a Ramsey bound.

8.4.1 Minimization Variant

We start with the minimization variant which turns out to be easier than the maximization variant. This is most likely because of the following bound on t.

Lemma 8.31. *Let (G, k, t) be an instance of* Min α-FCGP. *If $t \geq dk$, then (G, k, t) is a trivial yes-instance..*

Proof. If $|V(G)| < k$, then we have a trivial no-instance and the statement follows immediately. Hence, in the following we assume that $|V(G)| \geq k$. Next, we show that there exists a solution S such that $|F| \leq dk$ where F is the set of edges with at least one endpoint in S. Then, it follows that $\text{val}(S) \leq t$ in such instances of Min α-FCGP and the statement is proven. Let σ be a degeneracy ordering of G and let $S \coloneqq \{v_1, \ldots, v_k\}$ consist of the first k vertices of σ. Each edge in F is of the form xy where $x < y$ with respect to σ. Since S consists of the first k vertices of σ and G is d-degenerate, we conclude that for $x = v_i$ with $i \in [k]$ there are at most d edges in F of the form xy. Since $|S| = k$, F contains at most dk edges. □

Shachnai and Zehavi [209] showed that Min α-FCGP with $\alpha \in (0, 1]$ admits an FPT-algorithm with respect to $k + t$. Hence, we obtain the following.

Corollary 8.32. Min α-FCGP *for $\alpha > 0$ is FPT parameterized by $d + k$.*

Naturally, we may now ask whether this FPT result can be strengthened to a polynomial kernel. As shown by Theorem 8.19, the non-degrading case of Min α-FCGP does not admit a polynomial kernel even on graphs with constant maximum degree which implies constant degeneracy. In contrast, the degrading variant has a kernel whose size is polynomial in $d + k$.

Theorem 8.33. Degrading Min α-FCGP *admits a kernel of size $(d + k)^{\mathcal{O}(1)}$.*

First, we consider the case $\alpha = 0$. Recall that Sparsest k-Subgraph is the special case of α-FCGP for minimization and $\alpha = 0$.

Proposition 8.34. Sparsest k-Subgraph *admits a kernel with* $\mathcal{O}(dk)$ *vertices and of size* $\mathcal{O}(d^2k)$.

Proof. Let (G, k, t) be an instance of Sparsest k-Subgraph. Assume that $|V(G)| \geq dk$. Now, since G is d-degenerate, we obtain an independent set I of size at least k in G. Observe that $\mathrm{val}(S) = 0$ and thus (G, k, t) is a trivial yes-instance. Hence, $|V(G)| < dk$. Since the number of edges in a d-degenerate graph is bounded by $d \cdot |V(G)|$ the statement follows. □

Now, we consider $\alpha \in (0, 1/3)$.

Lemma 8.35. Min α-FCGP *for* $\alpha \in (0, 1/3)$ *admits a kernel of size* $\mathcal{O}(d^4k^5)$.

Proof. Let I be an instance of Min α-FCGP for a fixed $\alpha \in (0, 1/3)$. First, we transform I into an equivalent instance $I' \coloneqq (G, \emptyset, \mathsf{counter}, k, t)$ of Annotated Min α-FCGP with parameter α. Note that the upper bound for t from Lemma 8.32 still holds for I'. Observe that if G contains a vertex v such that $\alpha \deg^{+c}(v) \geq t + k$, then we have for each solution S containing v that $\mathrm{val}(S) > t$. Hence, if there is a solution S for I', then $v \notin I$. Thus, it is safe to apply the Exclusion Rule (Reduction Rule 8.2) to vertex v. In the following, we assume that the Exclusion Rule (Reduction Rule 8.2) is applied exhaustively.

Note that now we have $\Delta(G) < (t + k) \cdot \alpha^{-1}$ and also $\Gamma(G) < (t + k) \cdot \alpha^{-1}$. Next, we can apply Reduction Rule 8.3 exhaustively. Analogously to the proof of Proposition 8.11, after the exhaustive application of Reduction Rule 8.3 we have $|V(G)| \leq \Delta k$. Afterwards, with the reduction described in Lemma 8.9, we can construct an equivalent instance (G', k, t'') of Min α-FCGP of size $\mathcal{O}((\Delta(G) + \Gamma(G))^3 |V(G)|) = \mathcal{O}(d^4k^5)$. □

Now, Proposition 8.34 and Lemma 8.35 lead to Theorem 8.33.

8.4.2 Maximization Variant

Recall that MaxPVC is the special case of Max α-FCGP with $\alpha = 1/2$. Amini et al. [8] showed that MaxPVC can be solved in $\mathcal{O}^*((dk)^k)$ time. Adapting this algorithm leads to an FPT-algorithm for α-FCGP with respect to $d + k$ for $\alpha \neq 0$. The main distinction is that in our adaptation it is important to use the annotated variant to keep track of vertices being contained in a partial solution.

Proposition 8.36. Degrading α-FCGP *can be solved in* $\mathcal{O}^*((dk)^k)$ *time for* $\alpha \neq 0$.

Proof. Note that it is sufficient to present an algorithm with running time $\mathcal{O}^*((dk)^k)$ for Annotated Degrading α-FCGP for $\alpha \neq 0$ since each instance (G, k, T) of Degrading α-FCGP can be trivially transformed into an equivalent instance $(G, \emptyset, \mathsf{counter}, k, t)$ of Annotated α-FCGP. We present a search-tree algorithm with depth at most k such that each node has at most dk children. The root of the search-tree corresponds to the instance $(G, \emptyset, \mathsf{counter}, k, t)$.

Now, we consider the instance $I \coloneqq (G, T, \mathsf{counter}, k', t')$ instance of Annotated Degrading α-FCGP with $\alpha \neq 0$. Let $L \coloneqq \{v \in V(G) \setminus T \mid \mathrm{cont}(v, T) \geq t'/k'\}$ (maximization) and $L \coloneqq \{v \in V(G) \setminus T \mid \mathrm{cont}(v, T) \leq t'/k'\}$ (minimization). Clearly, if $|L| \geq dk$, then there exists a subset $I \subseteq L$ such that I induces an independent set in G. For maximization we conclude from $\mathrm{cont}(v) \geq t'/k'$ for each vertex $v \in I$ we that $\mathrm{val}(I) \geq t'$. Otherwise, $|L| < dk$ and for minimization we conclude from $\mathrm{cont}(v) \leq t'/k'$ for each vertex $v \in I$ we that $\mathrm{val}(I) \leq t'$. Otherwise, $|L| < dk$. By definition of L we have $\mathrm{cont}(u) < t'/k'$ (maximization) and $\mathrm{cont}(u) > t'/k'$ (minimization) for each $u \in V(G) \setminus (L \cup T)$. Thus, k vertices from $V(G) \setminus (L \cup T)$ are not sufficient to obtain value at least (maximization) or at most (minimization) t', Hence, each solution contains at least one vertex of L. In other words, I is a yes-instance if and only if at least one of the instance $(G, T \cup \{v\}, \mathsf{counter}, k' - 1, t' - \mathrm{cont}(v)$ for some $v \in L$ is a yes-instance. Hence, each node in the enumeration tree has at most dk children. Furthermore, since in each child of I the parameter k' is reduced by 1, the depth is bounded by k. Hence, we obtain an $\mathcal{O}^*((dk)^k)$ for Annotated α-FCGP. □

The rest of this section is devoted to the proof of the next theorem.

Theorem 8.37. Degrading Max α-FCGP *admits a kernel of size* $k^{\mathcal{O}(d)}$ *but, unless* coNP $\subseteq$ NP/poly*, no kernel of size* $\mathcal{O}(k^{d-2-\epsilon})$.

In particular, this implies that MaxPVC admits a kernel of size $k^{\mathcal{O}(d)}$. We remark that a compression of size $(dk)^{\mathcal{O}(d)}$ was obtained independently by Panolan and Yaghoubizade [187].

A kernel for biclique-free graphs in the degrading case. We next develop a kernelization algorithm with the size bound $k^{\mathcal{O}(d)}$. In fact, our algorithm works for *biclique-free graphs*—graphs that do not have a biclique $K_{a,b}$ as a subgraph for $a \leq b \in \mathbb{N}$. Note that a d-degenerate graph has no $K_{d+1,d+1}$ as a subgraph, since every vertex in a $K_{d+1,d+1}$ has degree $d + 1$.

Note that a clique of size $a + b$ contains $K_{a,b}$ as a subgraph. So given a graph G with no occurrence of $K_{a,b}$ on at least $\binom{a+b+k-2}{k-1} \in k^{\mathcal{O}(a+b)}$ vertices, one can find

an independent set of size k in polynomial time (see Section 2.1). We show that this upper bound on the number of vertices can be improved: the sum $a + b$ in the exponent can be replaced by $\min\{a, b\}$.

Lemma 8.38. *For $a \leq b \in \mathbb{N}$, let G be a graph that contains no $K_{a,b}$ as a subgraph. If G has at least $R(k)$ vertices, then we can find in polynomial time an independent set of size k, where $R(k) \in (a+b)^{\mathcal{O}(a)} \cdot k^a$*

Proof. We first show that if G has at least $k+b\binom{k}{a}+\sum_{\ell\in[a-1]} R(a+b,\ell+1)\binom{k}{\ell}$ vertices, then it contains an independent set of size k. Afterwards, we give a polynomial-time algorithm to find such an independent set of size k. Let I be a maximum independent set in G. We assume for contradiction that $|I| < k$. We prove that there are at most $t\binom{k}{a}$ vertices that have at least a neighbors in I and that there are at most $\sum_{\ell\in[a-1]} R(a+b,\ell+1)$ vertices that have at most $a-1$ neighbors in I.

For each subset $X \subseteq I$ of size exactly a, note that there are at most b vertices v such that $N(v) \supseteq X$, since otherwise there is a $K_{a,b}$ in G. It follows that the number of vertices with at least a neighbors in I is at most $t\binom{|I|}{a} \leq b\binom{k}{a}$. Consider a set $X \subseteq I$ of size $\ell \in [a-1]$. Let $V_X \coloneqq \{v \in V(G) \setminus I \mid N(v) \cap I = X\}$. Then, there is no independent set I' of size $\ell+1$ in V_X, since otherwise $(I \setminus X) \cup I'$ is an independent set of size at least $|I|+1$, contradicting the fact that I is an independent set of maximum size. Moreover, there is no clique of size $a+b$ in V_X. Thus, $|V_X| < R(a+b,\ell+1)$. The number of vertices with at most $a-1$ neighbors in I is then at most $\sum_{X\subseteq I,|X|=\ell\in[a-1]} R(a+b,\ell+1)\binom{|I|}{\ell} \leq \sum_{\ell\in[a-1]} R(a+b,\ell+1)\binom{k}{\ell}$.

We turn the argument above into a polynomial-time algorithm as follows. Suppose that we have an independent set I' of size smaller than k. As discussed above, there are at most $b \cdot \binom{k}{a}$ vertices that have at least s neighbors in I'. Hence, there is a vertex set $X \subseteq I'$ of size ℓ such that $|V_X| > R(a+b,\ell+1)$. Note that X can be found in polynomial time, for instance, by counting the number of vertices v' such that $N(v') \cap I' = N(v) \cap I'$ for each vertex $v \in V(G)$. We can then find an independent set I'' of size $\ell+1$ in X (this can be done in polynomial time as discussed in Section 2.1). This way, we end up with an independent set $(I' \setminus X) \cup I''$ of size at least $|I'|+1$. Note that this procedure of finding an independent set of greater size is repeated at most k times, and thus the overall running time is polynomial. □

We remark that for fixed $a \leq b \in \mathbb{N}$, Lemma 8.38 gives us an $\mathcal{O}(n^{1-1/a})$-approximation algorithm for Independent Set that runs in n^ℓ time for some constant ℓ not depending on a or b. An $\mathcal{O}(n^{1-1/a})$-approximation algorithm is known on graphs where $K_{a,b}$ is excluded as an *induced subgraph* [30, 64]. However, these algorithms have running time $n^{\Omega(a)}$.

We now apply Lemma 8.38 to obtain a lemma analogous to Lemma 8.22.

Lemma 8.39. *Suppose that $\Delta \geq R(bk^{a-1})$. Then, we can find in polynomial time a set X of $i \in [a-1]$ vertices and an independent set I with the following properties:*

1. *The set $I \subseteq \bigcap_{x \in X} N(x)$ is an independent set of size at least $bk^{a-i} + 1$.*
2. *For every vertex $u \in V(G) \setminus X$, it holds that $|N(u) \cap I| \leq bk^{a-i-1}$.*

Proof. Let v be a vertex with $\deg(v) \geq R(bk^{a-1})$. By Lemma 8.38, there is an independent set I_v of size bk^{b-1} in $N(v)$ (which can be found in polynomial time). Let X be an inclusion-wise maximal set of i vertices containing v with $|\bigcap_{x \in X} N(x) \cap I_v| > bk^{a-i}$. Such a set can be found by the following polynomial-time algorithm: We start with $X = \{v\}$ and $i = 1$. We will maintain the invariant that $|X| = i$. If there exists a vertex $v' \in V(G) \setminus X$ with $|N(v') \cap \bigcap_{x \in X} N(x) \cap I_v| > bk^{a-i-1}$, then we add v' to X and increase i by 1. We keep doing so until there remains no such vertex v'.

We show that this algorithm terminates for $i = |X| \leq a - 1$. Assume to the contrary that the algorithm continues for $i = a - 1$. We then have that $|N(v') \cap \bigcap_{x \in X} N(x) \cap I_v| > bk^{a-i-1}$ for some vertex $v' \in V(G) \setminus X$. It follows that the set $X \cup \{v'\}$ (which is of size a) has more than b common neighbors, contradicting the fact that G has no $K_{a,b}$ as a subgraph.

Finally, we show that the set X found by this algorithm and $I \coloneqq \bigcap_{x \in X} N(x) \cap I_v$ satisfy the three properties of the lemma. We have $|I| = |\bigcap_{x \in X} N(x) \cap I_v| = |N(v') \cap \bigcap_{x \in X \setminus \{v'\}} N(x) \cap I_v| > bk^{a-(i-1)-1} = bk^{a-i}$, where v' is the last vertex added to X. Moreover, since X is inclusion-wise maximal, we have $|N(u) \cap I| = |N(u) \cap \bigcap_{x \in X} N(x) \cap I_v| \leq bk^{a-i-1}$ for every vertex $u \in V(G) \setminus X$. □

Reduction Rule 8.9. *Let $\mathcal{I}$ be an instance of* Annotated Degrading α-FCGP. *Let X, I be as specified in Lemma 8.39 and let $v \in I$ be a vertex such that every other vertex in I is better than v. Then, apply the Exclusion Rule (Reduction Rule 8.2) to v.*

Lemma 8.40. *Reduction Rule 8.9 is correct.*

Proof. We show the proof for the maximization variant; the minimization variant follows analogously. For the sake of contradiction, assume that every solution S contains v. By Lemma 8.39, every vertex $u \in V(G) \setminus X$ has at most bk^{a-i} neighbors in I. Moreover, since I is an independent set, we have $|I \cap N[v']| = 1$ for every vertex $v' \in I$ (including v). For $S' \coloneqq S \setminus X$, we have

$$\begin{aligned} |I \setminus N[S']| &\geq |I| - |I \cap N[v]| - |I \cap N[S' \setminus \{v\}]| \\ &\geq (bk^{a-i} + 1) - (k-1)bk^{a-i-1} - 1 = bk^{a-i-1} > 0. \end{aligned}$$

Let v' be an arbitrary vertex in $I \setminus N[S']$. We show that $\mathrm{cont}(v', S \setminus \{v\}) \geq \mathrm{cont}(v, S \setminus \{v\})$. By Lemma 8.6, this would imply that $(S \setminus \{v\}) \cup \{v'\}$ is a solution not containing v. Since v and v' are both adjacent to all vertices of X and $\alpha \in (1/3, 1]$, we have

$$\begin{aligned}\mathrm{cont}(v', S \setminus \{v\}) &= \alpha \deg^{+c}(v') + (1 - 3\alpha)|X \cap (S \setminus \{v\})| \\ &\geq \alpha \deg^{+c}(v) + (1 - 3\alpha)|X \cap (S \setminus \{v\})| \\ &\geq \alpha \deg^{+c}(v) + (1 - 3\alpha)|N(v) \cap (S \setminus \{v\})| = \mathrm{cont}(v, S \setminus \{v\}).\end{aligned}$$

Here, the first inequality follows from the fact that v' is better than v. □

By applying Reduction Rule 8.9 exhaustively, we end up with an instance with maximum degree $\Delta \leq R(bk^{a-1})$. The following proposition then follows from Proposition 8.18 using the bound in Lemma 8.38:

Proposition 8.41. *For any $a \leq b \in \mathbb{N}$,* Degrading α-FCGP *on graphs that do not contain $K_{a,b}$ as a subgraph has a kernel of size $(R(bk^{a-1}) + k)^{\mathcal{O}(1)} \in b^{\mathcal{O}(a)} k^{\mathcal{O}(a^2)}$.*

Note that a d-degenerate graph contains no $K_{d+1,d+1}$ as a subgraph. Thus, we obtain a kernel of size $k^{\mathcal{O}(d^2)}$ for fixed d. In fact, we obtain a smaller kernel using the folklore fact that any d-degenerate graph on at least $(d + 1)k$ vertices has an independent set of size k.

Lemma 8.42. Degrading α-FCGP *admits a kernel of size $k^{\mathcal{O}(d)}$.*

Proof. Since a d-degenerate graph has no $K_{d+1,d+1}$ as a subgraph, there is a kernel of size $(R(dk^{d-1}) + k)^{O(1)}$ by Proposition 8.41. Recall that for $\ell \in \mathbb{N}$, $R(\ell)$ denotes an integer such that any graph on $R(\ell)$ vertices has an independent set of size ℓ. Since $R(\ell) \leq (d+1)\ell$ for any d-degenerate graphs, there is a kernel of size $(R(dk^{d-1}) + k)^{O(1)} \in k^{\mathcal{O}(d)}$. □

A matching Lower Bound. In the following, we show that significant improvement in Lemma 8.42 is unlikely. This, together with Lemma 8.42, implies Theorem 8.37.

Proposition 8.43. Degrading Max α-FCGP *admits no kernel of size $\mathcal{O}(k^{d-2-\epsilon})$ unless* coNP $\subseteq$ NP/poly.

Proof. We provide a weak q-composition from Independent Set on 2-degenerate graphs to Degrading Max α-FCGP in $(q+2)$-degenerate graphs. Here, we assume that $k > |c(\alpha)|/(3\alpha - 1)$ for some constant $c(\alpha) > 0$ which will be specified below.

Let $[t]^q$ be the set of q-dimensional vectors whose entries are in $[t]$. For a vector $x \in [t]^q$ we denote by x_i the ith entry of x. Next, assume that $q \geq 2$ is a constant and that we are given exactly t^q instances $\mathcal{I}_x \coloneqq (G_x, k)$ of Independent Set on 2-degenerate graphs. We construct an equivalent instance (H, k', t') of Degrading Max α-FCGP as follows.

Construction: We add an independent set J (the *instance choice gadget*) consisting of tq vertices to H. The vertices of J are denoted by w^i_j with $i \in [q]$ and $j \in [t]$. Next, for each $x \in [t]^q$ we add the graph G_x to H. In other words, we added the *instance gadgets* to H. By D we denote the union of these vertices. Furthermore, for each $x \in [t]^q$ and for each vertex $v \in V_x$ we add the edge $vw^i_{x_i}$ for each $i \in [q]$. Now, we fix an integer $\ell > t^q \cdot n > k$. For each vertex w^i_j in the independent set D we add leaf vertices such that $\deg(w^i_j) = \ell + 1$. Next, for each vertex $v \in D$ we add leaf vertices such that $\deg(v) = \ell$. By L we denote the union of all these added leaf vertices. Finally, we set $k' \coloneqq k + tq - q$ and $t' \coloneqq \alpha\,[k\ell + (tq - q)(\ell + 1)]$.

Degeneracy: Next, we show that $d(H) = q + 2$. Let F be any subgraph of H. We have to show that F contains a vertex with degree at most $d(H) = q + 2$. Clearly, if F contains a leaf vertex of L, then F contains a vertex of degree 1. Thus, in the following we can assume that $F \cap L = \emptyset$. If $F \subseteq J$ then F is edgeless. Hence, let $F^D \coloneqq V(F) \cap D \neq \emptyset$ and let $F^D_x = F^D \cap V_x \neq \emptyset$ for some $x \in [t]^q$. Since by assumption $\mathcal{I}_x$ is 2-degenerate, there exists a vertex $v \in F^D_x$ such that $|N(v) \cap F^D_x| \leq 2$. By construction v has only neighbors in V_x and exactly q neighbors in J an no neighbors in V_y for some $y \neq x$. Thus, v has degree at most $q + 2$.

Correctness: In the following, we prove that there exists an independent set of size exactly k for some instance $\mathcal{I}_x$ with $x \in [t]^q$ if and only if there exists a vertex set S of size exactly k' in H such that $\mathrm{val}(S) \geq t'$.

Suppose that instance $\mathcal{I}_x$ has an independent set I of size exactly k for some $x \in [t]^q$. By $J^* \coloneqq J \setminus \bigcup_{i \in [q]} \{w^i_{x_i}\}$ we denote the non-neighbors of V_x in J. Note that $|J^*| = tq - q$. We show that $S \coloneqq I \cup J^*$ is a solution of (H, k', t). Clearly, $|S| = k + tq - q = k'$. Since no vertex of I is connected with any vertex in J^*, I is an independent set, and since J^* is an independent set, we conclude that $I \cup J^*$ is an independent set. Thus, $E_H(S) = \emptyset$. Furthermore, since each vertex in I has degree ℓ, and since each vertex in J^* has degree $\ell + 1$, we conclude that $m_H(S, V(H) \setminus S) = k\ell + (tq - q)(\ell + 1)$. Thus, $\mathrm{val}(S) = t'$ and hence (H, k', t') is a yes-instance of Degrading Max α-FCGP.

Conversely, suppose that (H, k', t') has a solution $S \subseteq V(H)$ of size exactly k' with $\mathrm{val}(S) \geq t'$. First, we show that S cannot contain any leaf-vertex in L. Assume towards a contradiction that $S \cap L \neq \emptyset$. Let $v \in S \cap L$ and let $S' \coloneqq S \setminus \{v\}$. According to Lemma 8.26, $\mathrm{val}(S)$ is maximal if S' is an independent set and $E(v, S') = \emptyset$. Since the maximum degree in H is $\ell + 1$ and since $\deg(v) = 1$, we obtain that $\mathrm{val}(S) \leq \alpha\,[1 + (k'-1)(\ell+1)]$. Hence,

$$\begin{aligned} t - \mathrm{val}(S) &\geq \alpha k\ell + \alpha(tq-q)(\ell+1) - [\alpha + \alpha k\ell + \alpha k + \alpha(tq-q)(\ell+1) - \alpha(\ell+1)] \\ &= \alpha(\ell - k) > 0 \end{aligned}$$

since $\ell > k$. Hence, in the following we can assume that $S \cap L = \emptyset$. Let $J^* \coloneqq J \cap S$, $|J^*| = z$, and $D^* \coloneqq S \setminus J^* \subseteq D$. In the following, we show that $z = (t-1)q$ and that there exists an $x \in [t]^q$ such that $N(V_x) \cap J^* = \emptyset$. Therefore, we consider the cases that $z < tq - q$ and that $z > tq - q$. In both cases we verify that $\mathrm{val}(S) < t'$ for each solution with exactly z vertices in J.

Case 1: $z \leq tq-q-1$. Let $p \coloneqq tq-q-z$. Note that $p \in [tq-q]$. Recall that $\deg(v) = \ell$ for each vertex $v \in D$. Thus, by Lemma 8.26, $\mathrm{val}(S)$ is maximized if $m_H(D^*) + m_H(J^*, D^*)$ is minimized. Since $|J^*| < tq - q$, it is possible that no vertex of D^* is adjacent to any vertex in J^*. Thus, $\mathrm{val}(S)$ is maximal if S is an independent set with exactly $tq - q - p$ vertices in J. Hence, $\mathrm{val}(S) \leq \alpha\,[(k+p)\ell + (tq-q-p)(\ell+1)]$. Now, we obtain that

$$\begin{aligned} t - \mathrm{val}(S) &\geq \alpha k\ell + \alpha(tq-q)(\ell+1) - [\alpha k\ell + \alpha p\ell + \alpha(tq-q)(\ell+1) - \alpha p(\ell+1)] \\ &= \alpha p > 0. \end{aligned}$$

Thus, $t' - \mathrm{val}(S) > 0$, a contradiction.

Case 2: $z \geq tq-q+1$. Let $p \coloneqq z-tq+q$. In other words $p \in [q]$. By the pigeonhole principle there exist at least p indices $i \in [q]$ such that $w^i_j \in S$ for each $j \in [t]$. Recall that by construction, each vertex $v \in D$ has exactly one neighbor in the set $\{w^i_j, j \in [t]\}$. Since $|D^*| = k - p$ we conclude that $m_H(J^*, D^*) \geq (k-p)p$. Recall that $\deg(v) = \ell$ for each vertex $v \in D$. Thus, by Lemma 8.26, $\mathrm{val}(S)$ is maximized if $m_H(D^*) + m_H(J^*, D^*)$ is minimized. Hence, $\mathrm{val}(S)$ is maximal if $m_H(J^*, D^*) = (k-p)p$ and D^* is an independent set. Since $H[J]$ is an independent set we obtain that

$$\mathrm{val}(S) \leq (1-\alpha)(k-p)p + \alpha\,[(k-p)(\ell-p) + (t-1)q(\ell+1) + p(\ell+1-k+p)]\,.$$

Now, we obtain that

$$\begin{aligned} t - \mathrm{val}(S) &\geq \alpha k\ell + \alpha(tq-q)(\ell+1) - kp \\ &\quad + p^2 + 3\alpha kp - 3\alpha p^2 - \alpha k\ell - \alpha(tq-q)(\ell+1) - \alpha p \\ &= -kp + p^2 + 3\alpha kp - 3\alpha p^2 - \alpha p \\ &= (3\alpha-1)kp + (1-3\alpha)p^2 - \alpha p. \end{aligned}$$

From $p \geq 0$ and $3\alpha - 1 > 0$ for $\alpha \in (1/3, 1]$, we obtain that

$$t - \mathrm{val}(S) \geq (3\alpha-1)k + (1-3\alpha)p^2 - \alpha p = (3\alpha-1)k + c'(\alpha, p)$$

where $c'(\alpha, p) < 0$ is a constant only depending on the parameters α, and p. We set $c(\alpha) \coloneqq \max_{p\in[q]} c'(\alpha, p)$ which is a constant smaller than 0 only depending on α. In other words, $t - \mathrm{val}(S) \geq (3\alpha-1)k + c(\alpha) > 0$ since by assumption $k > |c(\alpha)|/(3\alpha-1)$. A contradiction to the fact that $\mathrm{val}(S) \geq t'$.

Hence, $|J^*| = tq - q$ and thus $|D^*| = k$. According to Lemma 8.26, $\mathrm{val}(S)$ is maximal if $m_H(D^*) + m_H(J^*, D^*)$ is minimal. Observe that if $E_H(J^*, D^*) = \emptyset$ and D^* is an independent set, then $\mathrm{val}(S) = t'$. Otherwise, if $E_H(J^*, D^*) \neq \emptyset$ or if D^* is no independent set, then $\mathrm{val}(S) < t'$. Thus, $E_H(J^*, D^*) = \emptyset$ and D^* is an independent set. Now, if there exist two vertices $u, v \in D^*$ such that $u \in V_x$ and $v \in V_y$ with $x \neq y$ for $x, y \in [t]^q$, then $(N(u) \cup N(v)) \cap J \geq q+1$. Since $|J^*| = tq-q$ this implies that $E_H(J^*, D^*) \neq \emptyset$, a contradiction. Hence, $D^* \subseteq V_x$ for some $x \in [t]^q$. Furthermore, $J^* = J \setminus N(D^*)$. Thus, the instance x contains an independent set of size at least k.

Hence, we have a weak-q-composition from Independent Set to Degrading Max α-FCGP in $(d+2)$-closed graphs. Now, the proposition follows by Lemma 2.14. □

8.5 Parameterization by h-Index and Vertex Cover Number

To complete the picture of the parameterized complexity landscape, we consider two parameters that are larger than the degeneracy of G: the h-index of G and the vertex cover number of G.

Our results in this section are based on two data reduction rules. The first rule discards (according to the Exclusion Rule (Reduction Rule 8.2)) vertices with small contribution when there are sufficiently many vertices with high contribution. The

second rule adds vertices with very large contribution to a solution (according to the Inclusion Rule (Reduction Rule 8.1)) assuming there are only few vertices with large contribution. Below, we will specify when the contribution is small, or large.

Definition 8.44. Let $\mathcal{I}$ be an instance of Annotated α-FCGP and let $x \in \mathbb{N}$. Then $V_x \coloneqq \{v \in V(G) \mid \deg^{+c}(v) \geq x\}$.

This definition helps us to specify when the contribution is small, or large.

Lemma 8.45. *Let $\mathcal{I}$ be a yes-instance of* Annotated Max α-FCGP, *let $x \in \mathbb{N}$ with $|V_x| \geq k$, and let $v \in V(G)$ be a vertex with $\alpha \cdot \deg^{+c}(v) < \alpha x - |(1-3\alpha)k|$. Then, there is a solution S with $v \notin S$.*

Proof. Assume towards a contradiction that v is contained in each solution S. Observe that $v \notin V_x$. Since $|S| = k$ it follows that there is a vertex $u \in V_x \setminus S$. We claim that $S' \coloneqq (S \setminus \{v\}) \cup \{u\}$ is also a solution. This follows from Lemmas 8.6 and 8.7. □

Lemma 8.46. *Let $\mathcal{I}$ be a yes-instance of* Annotated Max α-FCGP, *let $x \in \mathbb{N}$ with $|V_x| \leq k$, and let $v \in V(G)$ be a vertex with $\alpha \deg^{+c}(v) \geq \alpha x + |(1-3\alpha)k|$. Then, there is a solution S with $v \in S$.*

Proof. Assume towards a contradiction that v is not contained in any solution S. Since $|S| = k$, $v \in V_x \setminus S$, and $|V_x| \leq k$ it follows that there is a vertex $u \in S \setminus V_x$. We claim that $S' \coloneqq (S \setminus \{u\}) \cup \{v\}$ is also a solution. This follows from Lemmas 8.6 and 8.7. □

8.5.1 Parameterization by h-Index

We start with the maximization variant and the h-index. As Non-Degrading Max α-FCGP does not admit a polynomial kernel with respect to k even if Δ is constant (see Theorem 8.19), the same holds for the h-index. We show that in contrast, the degrading case admits a polynomial kernel.

Proposition 8.47. Degrading Max α-FCGP *admits a size $\mathcal{O}(\alpha^{-2}(h^2k^2 + k^4))$ kernel.*

To show this result we make use of the two rules discarding (Lemma 8.45) and adding (Lemma 8.46) vertices with small or large contribution, respectively.

Lemma 8.48. *Let $\mathcal{I} = (G, k, t)$ be an instance of* Max α-FCGP.

1. *If $\alpha > 0$ and there are at least k vertices with degree at least $h+1+|(1-3\alpha)k \cdot \alpha^{-1}|$, then an equivalent instance of size $\mathcal{O}(h^2 + \alpha^{-1}hk^2)$ can be computed in polynomial time.*

2. *If $\alpha > 1/3$ and there are less than k vertices with degree at least $h + 1 + |(1-3\alpha)k \cdot \alpha^{-1}|$, then an equivalent instance of size $\mathcal{O}(\alpha^{-2}(h^2k^2 + k^4))$ can be computed in polynomial time.*

Proof. First, we transform $\mathcal{I}$ into an equivalent instance $(G, \emptyset, \mathsf{counter}, k, t)$ of Annotated Max α-FCGP where $\mathsf{counter}(v) = 0$ for all $v \in V(G)$. Then V_{h+1} is the set of vertices of degree greater than h. By the definition of h-index, we have $|V_{h+1}| \leq h$. We set $Y \coloneqq V(G) \setminus V_{h+1}$ to be the vertices with degree at most h. Let $x \coloneqq h + 1 + |(1-3\alpha)k \cdot \alpha^{-1}| > h$ (recall $\alpha > 0$). Then, we have $V_x \subseteq V_{h+1}$. We will assume that $|V(G)| > k$.

(1): $|V_x| \geq k$ and $\alpha > 0$. For each vertex $v \in Y$, we have $\deg(v) = \deg^{+c}(v) \leq h < x - |(1-3\alpha)k \cdot \alpha^{-1}|$. Hence, by Lemma 8.45, there is a solution not containing v. We thus can apply the Exclusion Rule (Reduction Rule 8.2) on v. Since this application does not change $\deg^{+c}(u)$ for any $u \in Y$, we can apply the Exclusion Rule (Reduction Rule 8.2) on all vertices in Y to obtain a graph with h vertices (the vertices in V_{h+1}).

Removing annotations by Lemma 8.8 results in an instance whose size is bounded in terms of Δ and Γ. Thus, we need to bound these two parameters since every vertex not in V_{h+1} has been deleted. We clearly have $\Delta \leq |V(G)| = h$. To bound Γ, we apply the following procedure: We apply Reduction Rule 8.6 exhaustively throughout. We then end up with a vertex v with $\mathsf{counter}(v) = 0$. If there exists a vertex $u \in V(G)$ with $\mathsf{counter}(u) > \deg(v) + |(1-3\alpha)k \cdot \alpha^{-1}|$, then we can apply the Exclusion Rule (Reduction Rule 8.2) on u because u is strictly better than v by Lemma 8.7. After this procedure, we may assume that $\Gamma \leq h + |\alpha^{-1} - 3|k$. By Lemma 8.8, we obtain an equivalent instance of size $\mathcal{O}((\Delta + \Gamma + \alpha^{-1}) \cdot |V(G)|) = \mathcal{O}(h^2 + \alpha^{-1}hk)$.

(2): $|V_x| < k$ and $\alpha > 1/3$. Consider the set $V_{x+|(1-3\alpha)k \cdot \alpha^{-1}|}$ of vertices with degree more than $h + 2|(1-3\alpha)k \cdot \alpha^{-1}|$. By Lemma 8.46 for each $v \in V_{x+|(1-3\alpha)k \cdot \alpha^{-1}|}$ there is a solution containing v. Thus, we can apply the Inclusion Rule (Reduction Rule 8.1) on all vertices in $V_{x+|(1-3\alpha)k \cdot \alpha^{-1}|}$ to obtain an instance with $\Delta_{\overline{T}} \leq x + |(1-3\alpha)k \cdot \alpha^{-1}|$ (recall that $\Delta_{\overline{T}}$ is the maximum degree over vertices not in T). The exhaustive application of Reduction Rule 8.3 results in a graph with at most

$$\Delta_{\overline{T}}k + 1 \leq xk + |(1-3\alpha)k \cdot \alpha^{-1}|k + 1 = (h+1)k + \alpha^{-1}|2/\alpha - 6|k^2 + 1 = \mathcal{O}(hk + \alpha^{-1}k^2)$$

vertices. Since we are dealing with the degrading case, we can use Proposition 8.17 to obtain an instance for Max α-FCGP of size

$$\begin{aligned}\mathcal{O}(|V(G)|^2 + \alpha^{-1}|V(G)|k^2) &\subseteq \mathcal{O}((hk + \alpha^{-1}k^2)^2 + \alpha^{-1}(hk + \alpha^{-1}k^2)k^2) \\ &= \mathcal{O}(\alpha^{-2}(h^2k^2 + k^4)).\end{aligned}$$

Thus, the statement follows. □

Lemma 8.48 implies Proposition 8.47 and thereby the existence of polynomial kernel for $\alpha > 1/3$: apply (1) if k vertices have degree at least $h+1+|(1-3\alpha)\cdot\alpha^{-1}|$ and (2) otherwise. It is unlikely that Lemma 8.48 (2) can be extended to cover the case $\alpha \in (0, 1/3)$. This would imply that Non-Degrading Max α-FCGP admits a polynomial kernel with respect to $k+h$, contradicting Theorem 8.19 (which states that Non-Degrading Max α-FCGP does not admit a polynomial kernel with respect to k for constant Δ).

We complement this with showing fixed-parameter tractability for $k + h$.

Proposition 8.49. Non-Degrading Max α-FCGP *is fixed-parameter tractable with respect to* $k + h$.

Proof. We first transform $\mathcal{I}$ into an equivalent instance $(G, \emptyset, \mathsf{counter}, k, t)$ of Annotated Max α-FCGP where $\mathsf{counter}(v) = 0$ for all $v \in V(G)$. To make the description of the algorithm easier, we redefine val by

$$\mathrm{val}_G(S) \coloneqq \alpha(m(S, V(G) \setminus S)) + \mathsf{counter}(S) + (1-\alpha)m(S),$$

that is, we do not multiply the counter by α but instead add correct multipliers when updating the counter.

Now, let $V_{>h}$ be the set of vertices of degree at least $h+1$. For each subset T of size at most k of $V_{>h}$, branch into the case that $T = S \cap V_{>h}$. In each case, we may now apply the Exclusion Rule (Reduction Rule 8.2) to remove the vertices of $V_{>h} \setminus T$. Now the contribution of T depends on which vertices of $V(G) \setminus V_{>h}$ are contained in S. This can be incorporated into the counters of $V(G) \setminus V_{>h}$ as follows. Pick any vertex $u \in T$. Then, decrease t by $\mathsf{counter}(u) + (1-\alpha)|N(u) \setminus T| + \alpha|N(u) \cap T|$. Now, for each vertex $v \in N(u) \setminus T = N(u) \setminus V_{>h}$, add $(1-\alpha) - \alpha = 1 - 2\alpha$ to $\mathsf{counter}(v)$. The correctness of this step can be seen as follows: if v is not contained in S, then the contribution of uv is α and this contribution is already recorded in the decrease of t. However, if we also add v to u, then the contribution of uv is $1-\alpha$, so this gives a value of $1-\alpha$ for the internal edge uv minus α for the fact that uv is no longer an outgoing edge. Finally, remove u from G.

After this removal of vertices in T has been applied exhaustively, the remaining graph has only vertices of degree at most h. Our aim is to find in this graph a

vertex set S' of size $k - |T|$ that maximizes $\text{val}(S')$. Now this problem can be solved in $f(h,k) \cdot n^{\mathcal{O}(1)}$ time since val fulfills a property of fixed-cardinality optimization problems called *component linear* by Komusiewicz and Sorge [150]: First, $\text{val}(S \cup T) \leq \text{val}(S) + \text{val}(T)$ because an internal edge counts twice as much as an outgoing edge. Second, $\text{val}(S \cup T) \geq \text{val}(S) + \text{val}(T)$ if there are no edges between S and T in G.

Altogether, the running time is $h^k \cdot n^{\mathcal{O}(1)} \cdot f(h,k)$ since we create h^k many cases in the branching on $V_{>h}$. □

Minimization variant. Note that Degrading Min α-FCGP has a polynomial kernel with respect to $d + k$ (see Theorem 8.33) and, thus, also with respect to $h + k$. As Non-Degrading Min α-FCGP does not admit a polynomial kernel with respect to k even if Δ is constant (see Theorem 8.19), the same holds for the h-index.

8.5.2 Parameterization by Vertex Cover Number

We have shown that Max α-FCGP admits a polynomial kernel with respect to $h + k$ for $\alpha > 1/3$. For the larger parameter vertex cover number vc, we achieve a polynomial kernel for all $\alpha > 0$.

Proposition 8.50. *If $\alpha \neq 0$, then* Max α-FCGP *admits a kernel of size $\mathcal{O}(\mathsf{vc}(\mathsf{vc} + \alpha^{-1}k)^2)$.*

Proof. We follow the proof of Lemma 8.48. As $\mathsf{vc} \geq h$, we only need to extend the statement of Lemma 8.48 (2) concerning the case $\alpha \in (0, 1/3)$. Let (G,k,t) be an instance of Max α-FCGP. First, we transform (G,k,t) into an equivalent instance $(G, \emptyset, \mathsf{counter}, k, t)$ of Annotated Max α-FCGP where $\mathsf{counter}(v) = 0$ for all $v \in V(G)$. Then, let X be a vertex cover of size vc. We set $I \coloneqq V(G) \setminus X$ to be the independent set. Note that each vertex in I has degree at most vc. Moreover, we set $x \coloneqq \mathsf{vc} + |(1-3\alpha)k \cdot \alpha^{-1}| > \mathsf{vc}$ since $\alpha > 0$. Thus, $V_x \subseteq X$.

Case 1: $|V_x| \geq k$. This case follows from Lemma 8.48 (1).

Case 2: $|V_x| < k$. Consider the set $V_{x+|(1-3\alpha)k\cdot\alpha^{-1}|}$ of vertices with degree at least $\mathsf{vc} + 2|(1-3\alpha)k \cdot \alpha^{-1}|$. By Lemma 8.46, for each $v \in V_{x+|(1-3\alpha)k\cdot\alpha^{-1}|}$ there is a solution containing v. Thus, we can apply the Inclusion Rule (Reduction Rule 8.1) on every vertex in $V_{x+|(1-3\alpha)k\cdot\alpha^{-1}|}$ including it into T. (Recall that T is the set of vertices fixed in the solution.) We then have $\Delta_{\overline{T}} \leq x + |(1-3\alpha)k \cdot \alpha^{-1}|$. Thus, there are at most $\mathsf{vc} \cdot \Delta_{\overline{T}} \in \mathcal{O}(\mathsf{vc}^2 + \alpha^{-1}\mathsf{vc} \cdot k)$ vertices in I that have at least one

neighbor in $X \setminus T$. Denote these vertices by $I_{\overline{T}}$. The remaining vertices in $I \setminus I_{\overline{T}}$ have neighbors only in T. Hence, their contribution is fixed and we can get rid of all but the k vertices in $I \setminus I_{\overline{T}}$ with highest contribution using the Exclusion Rule (Reduction Rule 8.2). Thus, we are left with the vertices in X, in $I_{\overline{T}}$, and the k vertices with highest contribution of $I \setminus I_{\overline{T}}$. These are $\mathcal{O}(\mathsf{vc}^2 + \alpha^{-1}\mathsf{vc} \cdot k)$ many vertices. We remove the annotation using Lemma 8.8: we obtain an instance for Max α-FCGP of size $\mathcal{O}((\Delta_{\overline{T}} + \Gamma + \alpha^{-1})|V(G)| + \alpha^{-1}k|T|) = \mathcal{O}(\mathsf{vc}(\mathsf{vc} + \alpha^{-1}k)^2)$. □

For $\alpha = 0$, Max α-FCGP corresponds to Densest k-Subgraph and Clique is one of its special cases ($t = \binom{k}{2}$). Since Clique does not admit a polynomial kernel with respect to vc [25] (and any clique is of size at most $\mathsf{vc} + 1$), Densest k-Subgraph does not admit a polynomial kernel with respect to vc. However, Densest k-Subgraph can be solved by a straightforward algorithm in $\mathcal{O}^*(2^{\mathsf{vc}})$ time. Thus, Densest k-Subgraph admits a kernel of size $\mathcal{O}(2^{\mathsf{vc}})$.

Minimization variant. Recall that Degrading Min α-FCGP has a polynomial kernel with respect to $d + k$ (see Theorem 8.33) and thereby $\mathsf{vc} + k$. It remains to consider Non-Degrading Min α-FCGP parameterized by $\mathsf{vc} + k$.

Proposition 8.51. Min α-FCGP *admits a kernel of size* $\mathcal{O}((\alpha^{-2} + k)(\mathsf{vc} + \alpha^{-1}\mathsf{vc} \cdot k)^2)$ *for* $\alpha > 0$ *and of size* $\mathcal{O}(\mathsf{vc}^2 + \mathsf{vc} \cdot k)$ *for* $\alpha = 0$.

Proof. Let X be a vertex cover of size vc and let $I \coloneqq V(G) \setminus X$ be the independent set. Without loss of generality we can assume that $|I| \geq k$ since otherwise we already have a trivial $\mathsf{vc} + k$-vertex kernel. If $\alpha = 0$, we have a trivial yes-instance as $\mathrm{val}(I') = 0$ for all $I' \subseteq I$ of size k. Thus, in the following, we assume that $\alpha > 0$. Let (G, k, t) be an instance of Min α-FCGP. We transform (G, k, t) into an equivalent instance $(G, \emptyset, \mathsf{counter}, k, t)$ of Annotated Min α-FCGP where $\mathsf{counter}(v) = 0$ for all $v \in V(G)$.

Let $x \coloneqq \mathsf{vc} + |(1 - 3\alpha)k \cdot \alpha^{-1}|$. We first show that there is a solution that does not contain any vertex in V_x: To this end, observe that $\deg(v) \leq \mathsf{vc}$ for each $v \in I$. Hence, by Lemma 8.7, each vertex in I is strictly better than each vertex in V_x. Since $|I| \geq k$, it follows from Lemma 8.6 that there is a solution not containing any vertex from V_x. Hence, we can apply the Exclusion Rule (Reduction Rule 8.2) on each vertex in V_x. As a result, the remaining vertices in X form still a vertex cover and have degree less than $\mathsf{vc} + |(1 - 3\alpha)k \cdot \alpha^{-1}|$. Thus, less than $\mathsf{vc}(\mathsf{vc} + |(1 - 3\alpha)k \cdot \alpha^{-1}|)$ vertices in I have neighbors in X; the remaining vertices are isolated vertices. As the contribution of each isolated vertex v in any solution is exactly $\alpha \cdot \mathsf{counter}(v)$, we can simply sort the isolated vertices by their contribution and remove all but the k vertices with the

lowest contribution. Thus, we end up with at most $\mathsf{vc}+\mathsf{vc}(\mathsf{vc}+|(1-3\alpha)k\cdot\alpha^{-1}|)+k = \mathcal{O}(\mathsf{vc}^2+\alpha^{-1}\mathsf{vc}\cdot k)$ vertices. Using Lemma 8.9 to remove the annotation, we get an instance for MAX α-FCGP of size

$$\mathcal{O}(\alpha^{-2}(\Delta+\Gamma+k)^2+\alpha^{-1}(\Delta+\Gamma+k)\cdot|V(G)|) = \mathcal{O}((\alpha^{-2}+k)(\mathsf{vc}+\alpha^{-1}\mathsf{vc}\cdot k)^2)$$

Thus, the statement follows. □

8.6 Conclusion

We provided a systematic parameterized complexity analysis for α-FCGP (see Figure 8.2). Thereby, we answered the open question of Amini, Fomin, and Saurabh on whether MAXPVC (the special case of MAX α-FCGP for $\alpha = 1/2$) has a polynomial kernel in planar graphs in the affirmative. This open question was independently answered by Panolan and Yaghoubizade [187]. Furthermore, we also answered the open question of Kanesh et al. [125] on whether MAXPVC admits an FPT-algorithm with respect to $k+c$ in the affirmative. Arguably our most interesting observation is that MAXPVC and MAX $(k, n-k)$-CUT behave not only similarly in terms of fixed-parameter tractability and kernelization with respect to the parameters considered in this chapter but that these kernels can be obtained by the same algorithms.

We settled the existence of FPT-algorithms and polynomial kernels with respect to various parameters combined with the solution size k. Our polynomial kernels are, however, not optimized and thus the polynomials are of high degree. Looking for smaller kernels is thus an obvious first task. Furthermore, it is interesting to identify parameters for which the kernelization complexity of MAXPVC and MAX $(k, n-k)$-CUT behaves differently to see whether one of these problems is not just a reformulation of the second problem.

We studied the parameterized complexity of the standard parameter k plus one structural graph parameter. These additional graph parameters were maximum degree, c closure, degeneracy, h-index, and vertex cover number. More precisely, we presented a framework which lead to a polynomial kernel if the maximum degree is bounded by a function of the additional structural graph parameter. It would be interesting to see whether our framework can be exploited for other structural graph parameters as well. One such prominent example is the treewidth of the graph [199]. Another interesting parameter is the weak-γ-closure [84] which is a smaller parameter than the degeneracy and the c-closure. Obtaining a kernelization result for this parameter in the degrading case would directly strengthen our results for degeneracy (Theorem 8.37) and c-closure (Theorem 8.28). Our reduction rules for c-closure

(and degeneracy) cannot easily be adapted for the weak γ-closure for the following reason: For the kernel for the c-closure we showed that the maximal clique size is bounded in a polynomial only depending on $c+k$. We achieved this by showing that if a vertex w has sufficiently many neighbors which are better than w, then we can remove w (Reduction Rule 8.7). This argument fails for the weak closure γ since here the ordering of the vertices is important.

Furthermore, while we looked at parameters that are small in sparse graphs, can similar results be achieved for dense graphs as considered for example by Lochet et al. [159]?

Another interesting parameter to study is the budget t. The parameterized complexity for this parameter is already settled for the most prominent problems of α-FCGP. W[1]-hardness was shown for Densest k-Subgraph [62] and Sparsest k-Subgraph [62]. Furthermore, FPT-algorithms were provided for Maximum Partial Vertex Cover [22, 133], Minimum Partial Vertex Cover [133], Max $(k, n-k)$-Cut [29, 204], and Min $(k, n-k)$-Cut [55]. Hence, it is interesting to study the complexity of α-FCGP when $\alpha \notin \{0, 1/2, 1\}$. For the maximization variant, FPT for $\alpha > 0$ is trivial since $t \leq \Delta k$ in non-trivial instances and these problems admit an FPT-algorithm for $\Delta + k$ [29, 209]. Thus, it is interesting to study whether these problems admit a polynomial kernel with respect to t. A polynomial kernel for t for Max $(k, n-k)$-Cut (the special case of Max α-FCGP for $\alpha = 1$) was presented by Saurabh and Zehavi [204]. It would be interesting to see whether $\alpha = 1$ is the only case which leads to a polynomial kernel. For the minimization variant, similar arguments as for the maximization variant are not possible: here only instances with $t \leq \Delta k$ are non-trivial and existing algorithms for $\Delta + k$ [29, 209] do not imply an FPT-algorithm for t.

It would also be interesting to study the problem Connected α-FCGP. In this variant of α-FCGP we additionally require that the subgraph induced by the solution is connected. Saurabh and Zehavi studied this problem in the maximization variant with $\alpha = 1$, that is, Max $(k, n-k)$-Cut in which the solution has to be connected, under the name Multi-Node Hub (MNH) [205]. They noted that MNH is W[1]-hard with respect to the standard parameter solution size k and provided an FPT-algorithm for parameter t. Does MNH admit a polynomial kernel for t? It would be interesting to study the generalized problem Connected α-FCGP.

The W[1]-hardness of Connected α-FCGP for each fixed α with respect to k either follows directly by the existing reductions in the non-connected variant or by a simple adaption of these reductions, for example by adding a universal vertex and adapting the budget accordingly.

A first question is for which values of α Connected α-FCGP admits an FPT-

algorithm for t. Some hardness results follow by existing reductions which showed W[1]-hardness in the non-connected version: The W[1]-hardness for DENSEST k-SUBGRAPH for parameter $k + t$ even in 2-degenerate and 2-closed graphs of Raman and Saurabh also gives W[1]-hardness for this connected version since the constructed solution is connected. The parameterized complexity for all remaining cases of the maximization variant with $\alpha > 0$ and all cases for the minimization variant remain open. Similar to the non-connected variant, in the maximization case an FPT-algorithm for $\Delta + k$ would imply also an FPT-algorithm t since $t \geq \Delta k$ in non-trivial instances.

A second question is the parameterized complexity with respect to $\Delta + k$. Note that all these problems are FPT with respect to $\Delta + k$: The number of induced subgraphs of size k is bounded by $\mathcal{O}((e(\Delta - 1))^{(k-1)} \cdot (n/k))$ [28, Equation 7]. Hence, all possible solutions can be enumerated in FPT-time with respect to $\Delta + k$ and then the best induced subgraph can be outputted. Furthermore, a disjoint union of 2^p instances leads to an or-composition, showing that a polynomial kernel for $k + \Delta$ is not possible under standard assumptions. Hence, in order to obtain polynomial kernels other, bigger parameters than $k + \Delta$ have to be considered.

Furthermore, it would be interesting to verify experimentally the effectiveness of our framework to include vertices with "high" contribution into our solution and to remove vertices with "low" contribution. We think that applying these reduction rules exhaustively simplifies the input instances significantly and thus leads to fast implementations in practice. One reason for this is that in many social networks the degrees are distributed according to a power law [180], that is, there are few vertices with high degree and many vertices with low degree. With our framework we would put some vertices with high degree into our solution and would remove a huge number of vertices with small degree from the graph.

Chapter 9

Conclusion

In this thesis, we studied various (connected) subgraph problems. These problems have applications in community detection [40, 223] and in the identification of network motifs [127, 227].

In Chapter 3, we studied the problem of enumerating all connected induced subgraphs of size k. We improved upon the previous best delay due to Elbassioni [67] for enumerating all connected induced subgraphs. Furthermore, we provided an experimental evaluation of various algorithms for this task. In this thesis and also in many other works [4, 67, 111], the focus was on enumeration problems on classic graphs, so-called *static* graphs. Recently, some classic enumeration problems have also been studied in *temporal* graphs [113, 189]. Roughly speaking, in temporal graphs the edge set differs in various time steps. Temporal graphs are a popular tool to model applications with changing edge sets, for example different social contacts at different times during a week [169]. Also some enumeration problems have been studied in temporal graphs: for example it was shown that network motifs [189] and cliques [113] can also be efficiently enumerated in temporal graphs. Compared with static graphs the number of studied enumeration problems in temporal graphs is smaller. For future work, it is interesting to study further enumeration problems in temporal graphs. One example, is the study of enumerating connected subgraphs in temporal graphs.

Afterwards, in Chapter 4, we used the best algorithms of our study for enumerating all connected induced subgraphs of size k (see Chapter 3) as a foundation for our generic solver `FixCon` for Connected Fixed Cardinality Optimization (CFCO). For `FixCon` we provided several generic pruning rules as speed-ups. We showed that `FixCon` outperforms standard Integer Linear Programs (ILPs) for small values of k. We observed that `FixCon` scales much better with the graph size than the

standard ILPs, but the standard ILPs scale much better with the solution size than `FixCon`. In the future, one might add further generic pruning rules to `FixCon` for more speed-ups. With such additions, the above-mentioned advantages of `FixCon` could be extended further and then `FixCon` could become a fast universal tool for answering small queries, that is, asking for specific structures of k vertices, on big networks.

In Chapter 5, we studied several clique relaxations in terms of their (parameterized) complexity with respect to the parameter (weak) closure. The (weak) closure is a newly discovered graph parameter based on the triadic closure principle [84]. Our work extends previous works on clique relaxations parameterized by the maximum degree [111, 143] and the degeneracy [70, 110, 111, 143].

Currently, the parameter c-closure was only exploited for the design of FPT-algorithms [17, 125, 137, 140] (see also Chapters 5 and 8). One interesting route is to study the approximation of clique relaxations in c-closed graphs. One starting point could be the investigation of the approximation of Clique in c-closed graphs. In general, Clique does not have a factor $n^{1-\epsilon}$ approximation for each $\epsilon > 0$, unless P = NP [236]. Can this hardness result be extended to c-closed graphs or is there a poly(c)-approximation?

The closure number of a vertex is the maximum number of common neighbors with each other non-adjacent vertex. The closure number of a graph is the maximum among all closure numbers of the vertices and the weak closure number of a graph is the maximum among all minimal closure numbers of each induced subgraph of the graph. In other words, the closure behaves like the maximum degree (taking the maximum among all closure numbers) and the weak closure number behaves like the degeneracy (taking the maximum of the minimum of each induced subgraph). There is a parameter between the maximum degree and the degeneracy: the h-index (the largest number h such that at least h vertices have degree at least h). Now, one can define a closure parameter which behaves like the h-index. The *η-closure index* is the largest number η such that at least η vertices have closure number at least η. This parameter is smaller than the closure number c and larger than the weak closure number γ. In future work, one could study the parameterized complexity of classic graph problems with respect to η. More precisely, whenever one obtains an FPT-algorithm for c and W[1]-hardness for γ, one should study the complexity with respect to η to settle the structure which makes the problem hard.

Another new interesting research direction is the study of a distance to triviality parameter in the context of (weakly) closed graphs. Distance from triviality parameters have been studied, for example, for the maximum degree. More precisely, the *deletion distance to degree t* is the size of a smallest vertex set S such that $G - S$

has maximum degree t. A similar parameter for closure can be defined as follows: The *deletion distance to closure number* c is the size of a smallest vertex set S such that $G - S$ has closure number c. Note that this parameter is independent of the above-defined h-index variant of the closure number. In future work, one could study classic graph problems which are easy on c-closed graphs, that is, solvable in polynomial time or in FPT-time, with respect to this parameter.

In Chapter 6, we studied the (parameterized) complexity of three variants of s-Club (see Chapter 6). Two of these problems are motivated by the hub-and-spoke behaviour of the s-club model [110]. We studied these two problems with respect to the parameterized complexity of the standard parameter solution size k. Afterwards, in Chapter 7 we provided an implementation which outperforms the known ILP [3, 41] (only for one of these two problems an ILP is known).

In both of these models, only the structure of the solution S was important. In other words, the structure of the cut corresponding to S was irrelevant. For future work, it is interesting to also take this structure into account. For example, *isolated cliques* were studied, that is, clique variants such that the number of edges having exactly one endpoint in the clique is small [118, 144, 171]. This property can also be added to our model. If this isolation requirement is high and also each vertex is forced to be in many triangles, the detection of a vertex triangle s-club should become easier.

Moreover, in these two models, we focused on problems related to detecting a specific subgraph or a certain structure. It is interesting to study edge editing and deletion variants or vertex deletion variants of the problems studied in this thesis. For example, it is interesting to study variants of Vertex-Triangle s-Club in which one has to edit or delete edges, or delete vertices such that each connected component of the graph is a vertex triangle s-club. Until now, such variants have been studied for cliques [18, 210], for s-plexes [100, 221] and for 2-clubs [158]. In these problems one aims to cluster the input data, that is, one wants to classify similar objects or agents.

The third problem we studied in Chapter 6 is motivated by detecting communities containing a fixed set of vertices [124, 230]. We are not aware of a comprehensive study for clique relaxations with seeds. Hence, our work can be seen as a first step in this research direction. For future work it is thus interesting to study other clique relaxations with seed constraints. Seeded problems are also interesting in more general settings: Clique relaxations are special properties of vertex sets. A very essential class of properties are *hereditary* properties. In future work, one could study hereditary problems with seeds. It is known that these are exactly those properties which can be described with a forbidden subgraph characterization. Furthermore, it is

known that if the forbidden subgraph characterization is finite, then the corresponding vertex-deletion problem is solvable in FPT-time [37]. Clearly, the idea to branch on the forbidden subgraphs can also be applied in the setting with seeds: simply ignore the cases which correspond to the deletion of seeded vertices. The situation, however, changes if one considers hereditary properties with an infinite forbidden subgraph characterization: Then, no general FPT statement is known. For future work it is interesting to study hereditary problems with a infinite forbidden subgraph characterization with seeds. Can one identify properties which do behave differently in the sense of FPT and W-hardness in the variant with and without seeds?

Finally, in Chapter 8 we studied the parameterized complexity of a general graph partitioning problem generalizing many well-studied graph problems. We studied this generic problem parameterized by the solution size k plus one additional structural graph parameter like maximum degree Δ or c-closure. We provide a dichotomy into fixed-parameter tractable and W[1]-hard cases and we prove tight kernel bounds for these parameters. Previously, only FPT-algorithms for the general problem for $k + \Delta$ [29, 209] and results for special cases [22, 38, 74, 133, 187, 204] were known.

In our work, we only required that one side of the partition has size exactly k. In another variant of Max $(k, n-k)$-Cut there is the additional requirement that the partite set of size k is connected [205]. It is interesting to impose more of these requirements for the general graph partitioning problem, that is, one could aim to find a maximum $(k, n-k)$-cut of size at least t such that the side containing k vertices is an independent set or is a clique.

Finally, we discuss some general directions for future research. An essential question in theoretical computer science is how well theoretic results describe the success of implementations. For the problems investigated in this thesis, the theoretic results are not fully able to explain the speed of our implementations: For the enumeration of connected induced subgraphs our theoretical results explain that not too much time is spent in each branching tree node. For Vertex Triangle 2-Club, however, the situation is different: we showed that for each possible ℓ the problem can be solved in $2^{n-k}n^{\mathcal{O}(1)}$ time. This result does not explain why Vertex Triangle 2-Club can be solved much faster for large values of ℓ in real-world instances. One reason for this is that for large ℓ our reductions rules are applied more often, but this is not yet captured in the theoretic results. A subsequent question is: which structure (measured by some parameter) can explain the success of this rule? Thus, in future work one should try to detect these hidden structures to explain the success of our implementations. This mismatch between theory and implementations does not only exist for the problems studied in this thesis: For example, 2-Club implementations are very fast [109] but the theory only provides Turing kernels and

a similar $2^{n-k}n^{\mathcal{O}(1)}$ time algorithm which are not sufficient to explain their success.

Furthermore, in the design of FPT-algorithms the focus is often in making the factor $f(k)$ as small as possible and thereby accepting large polynomial factors. This might not be useful in practice: Even a quadratic factor in the input size might be too large if the graph is big. For example, for Vertex Triangle 2-Club the exhaustive application of our reduction rules was the most time-consuming part in our implementation. Thus, in future work one should further focus on designing linear-time FPT-algorithms, that is, algorithms with running time $f(k) \cdot n$. Such algorithms have been designed for example for Odd Cycle Transversal [119] and Dominating Set on planar graphs [220].

Finally, in this thesis we mainly focused on the parameter solution size k. Often, there is a lower bound for k. Recently, above-guarantee parameterizations for various problems have been studied [162, 177]. It is also interesting to follow this research direction for the problems in this thesis: for example, is Seeded s-Club still FPT for $k - |W|$, where W is the seed, or is Maximum Partial Vertex Cover still FPT for $t - q$ where q is the difference of the sum of the k vertices with highest degree and $\binom{k}{2}$ (this is a lower bound for the number of edges used twice)?

Bibliography

[1] Alexander A. Ageev and Maxim Sviridenko. Approximation algorithms for maximum coverage and Max Cut with given sizes of parts. In *Proceedings of the 7th International Conference on Integer Programming and Combinatorial Optimization (IPCO '99)*, volume 1610 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 1999. (Cited on p. 201)

[2] David L. Alderson. OR FORUM - catching the "network science" bug: Insight and opportunity for the operations researcher. *Operations Research*, 56(5):1047–1065, 2008. (Cited on p. 137)

[3] Maria Teresa Almeida and Raul Brás. The maximum l-triangle k-club problem: Complexity, properties, and algorithms. *Computers & Operations Research*, 111:258–270, 2019. (Cited on pp. 24, 137, 141, 175, 176, 177, 178, 180, 185, 191, 195, 196, 255)

[4] Mohammed Alokshiya, Saeed Salem, and Fidaa Abed. A linear delay algorithm for enumerating all connected induced subgraphs. *BMC Bioinformatics*, 20-S(12):319:1–319:11, 2019. (Cited on pp. 8, 40, 60, 75, 253, 285, 292)

[5] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. (Cited on pp. 80, 109)

[6] Ernst Althaus, Markus Blumenstock, Alexej Disterhoft, Andreas Hildebrandt, and Markus Krupp. Algorithms for the maximum weight connected k-induced subgraph problem. In *Proceedings of the 8th International Conference on Combinatorial Optimization and Applications (COCOA '14)*, volume 8881 of *Lecture Notes in Computer Science*, pages 268–282. Springer, 2014. (Cited on pp. 80, 106, 107)

[7] David Amar and Ron Shamir. Constructing module maps for integrated analysis of heterogeneous biological networks. *Nucleic Acids Research*, 42(7):4208–4219, 2014. (Cited on p. 22)

[8] Omid Amini, Fedor V. Fomin, and Saket Saurabh. Implicit branching and parameterized partial cover problems. *Journal of Computer and System Sciences*, 77(6):1159–1171, 2011. (Cited on pp. 200, 203, 236)

[9] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. (Cited on pp. 29, 222)

[10] Emmanuel Arrighi, Niels Grüttemeier, Nils Morawietz, Frank Sommer, and Petra Wolf. Multi-parameter analysis of finding minors and subgraphs in edge periodic temporal graphs. *CoRR*, abs/2203.07401, 2022. URL: `https://doi.org/10.48550/arXiv.2203.07401`, `arXiv:2203.07401`. (Cited on p. 8)

[11] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996. (Cited on pp. 36, 39, 40, 60)

[12] David A. Bader, Andrea Kappes, Henning Meyerhenke, Peter Sanders, Christian Schulz, and Dorothea Wagner. Benchmarking for graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining*, pages 73–82. Springer, 2014. (Cited on pp. 68, 86, 191)

[13] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, 2011. (Cited on p. 22)

[14] Balabhaskar Balasundaram, Sergiy Butenko, and Svyatoslav Trukhanov. Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization*, 10(1):23–39, 2005. (Cited on pp. 19, 24, 141, 176)

[15] Balabhaskar Balasundaram and F Mahdavi Pajouh. Graph Theoretic Clique Relaxations and Applications. *Handbook of combinatorial optimization*, pages 1559–1598, 2013. (Cited on p. 22)

[16] Baski Balasundaram. Cohesive Subgroup Model for Graph-Based Text Mining. In *Proceedings of the 4th International Conference on Automation Science and Engineering (CASE' 08)*, pages 989–994. IEEE, 2008. (Cited on p. 22)

[17] Balaram Behera, Edin Husic, Shweta Jain, Tim Roughgarden, and C. Seshadhri. FPT algorithms for finding near-cliques in c-closed graphs. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference, (ITCS 22)*, volume 215 of *LIPIcs*, pages 17:1–17:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on pp. 114, 254)

[18] Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297, 1999. (Cited on p. 255)

[19] Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Linear time subgraph counting, graph degeneracy, and the chasm at size six. In *Proceedings of the 11th Conference on Innovations in Theoretical Computer Science Conference (ITCS '20)*, volume 151 of *LIPIcs*, pages 38:1–38:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited on p. 77)

[20] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-Densities: an $O(n^{1/4})$ aapproximation for densest k-subgraph. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC '10)*, pages 201–210. ACM, 2010. (Cited on p. 200)

[21] Ginestra Bianconi, Richard K Darst, Jacopo Iacovacci, and Santo Fortunato. Triadic closure as a basic generating mechanism of communities in complex networks. *Physical Review E*, 90(4):042806, 2014. (Cited on p. 22)

[22] Markus Bläser. Computing small partial coverings. *Inf. Process. Lett.*, 85(6):327–331, 2003. (Cited on pp. 200, 250, 256)

[23] A. Blokhuis and A.E. Brouwer. Geodetic graphs of diameter two. *Geometriae Dedicata*, 25:527–533, 1988. (Cited on p. 125)

[24] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009. (Cited on p. 142)

[25] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014. (Cited on pp. 34, 203, 248)

[26] Vladimir Boginski, Sergiy Butenko, and Panos M. Pardalos. Mining market data: A network approach. *Computers & Operations Research*, 33(11):3171–3184, 2006. (Cited on pp. 19, 21, 199)

[27] Vladimir Boginski, Sergiy Butenko, Oleg Shirokikh, Svyatoslav Trukhanov, and Jaime Gil-Lafuente. A network-based data mining approach to portfolio selection via weighted clique relaxations. *Annals of Operations Research*, 216(1):23–34, 2014. (Cited on pp. 19, 21, 22)

[28] Béla Bollobás. *The Art of Mathematics – Coffee Time in Memphis*. Cambridge University Press, 2006. (Cited on pp. 20, 38, 251)

[29] Edouard Bonnet, Bruno Escoffier, Vangelis Th. Paschos, and Emeric Tourniaire. Multi-parameter analysis for local graph partitioning problems: Using greediness for parameterization. *Algorithmica*, 71(3):566–580, 2015. (Cited on pp. 10, 19, 200, 201, 202, 205, 250, 256)

[30] Édouard Bonnet, Stéphan Thomassé, Xuan Thang Tran, and Rémi Watrigant. An algorithmic weakening of the Erdős-Hajnal conjecture. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA '20)*, volume 173 of *LIPIcs*, pages 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited on p. 238)

[31] Jean-Marie Bourjolly, Gilbert Laporte, and Gilles Pesant. An exact algorithm for the maximum k-club problem in an undirected graph. *European Journal of Operational Research*, 138(1):21–28, 2002. (Cited on pp. 23, 141, 176)

[32] Robert A Briers. Incorporating connectivity into reserve selection procedures. *Biological Conservation*, 103(1):77–83, 2002. (Cited on p. 80)

[33] Markus Brill, Piotr Faliszewski, Frank Sommer, and Nimrod Talmon. Approximation algorithms for balancedCC multiwinner rules. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '19)*, pages 494–502. International Foundation for Autonomous Agents and Multiagent Systems, 2019. (Cited on p. 7)

[34] Coenraad Bron and Joep Kerbosch. Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–576, 1973. (Cited on p. 111)

[35] Maurizio Bruglieri, Matthias Ehrgott, Horst W. Hamacher, and Francesco Maffioli. An annotated bibliography of combinatorial optimization problems with fixed cardinality constraints. *Discrete Applied Mathematics*, 154(9):1344–1357, 2006. (Cited on p. 79)

[36] Nader H. Bshouty and Lynn Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS '98)*, volume 1373 of *Lecture Notes in Computer Science*, pages 298–308. Springer, 1998. (Cited on p. 200)

[37] Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. (Cited on p. 256)

[38] Leizhen Cai. Parameterized complexity of cardinality constrained optimization problems. *The Computer Journal*, 51(1):102–121, 2008. (Cited on pp. 25, 79, 80, 199, 200, 201, 202, 256)

[39] Leizhen Cai, Siu Man Chan, and Siu On Chan. Random separation: A new method for solving fixed-cardinality optimization problems. In *Proceedings of the Second International Workshop on Parameterized and Exact Computation (IWPEC '06)*, volume 4169 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2006. (Cited on pp. 20, 21, 79, 80, 81, 82, 109, 200)

[40] Peter J Carrington, John Scott, and Stanley Wasserman. *Models and Methods in Social Network Analysis*, volume 28. Cambridge university press, 2005. (Cited on pp. 21, 253)

[41] Filipa D. Carvalho and Maria Teresa Almeida. The triangle k-club problem. *Journal of Combinatorial Optimization*, 33(3):814–846, 2017. (Cited on pp. 24, 137, 141, 176, 255)

[42] Maw-Shang Chang, Li-Hsuan Chen, Ling-Ju Hung, Peter Rossmanith, and Guan-Han Wu. Exact algorithms for problems related to the densest k-set problem. *Information Processing Letters*, 114(9):510–513, 2014. (Cited on p. 200)

[43] Maw-Shang Chang, Ling-Ju Hung, Chih-Ren Lin, and Ping-Chen Su. Finding large k-clubs in undirected graphs. *Computing*, 95(9):739–758, 2013. (Cited on pp. 19, 24, 112, 141, 176)

[44] Zhengzhang Chen, William Hendrix, and Nagiza F. Samatova. Community-based anomaly detection in evolutionary networks. *Journal of Intelligent Information Systems*, 39(1):59–85, 2012. (Cited on p. 111)

[45] Zhengzhang Chen, Kevin A. Wilson, Ye Jin, William Hendrix, and Nagiza F. Samatova. Detecting and Tracking Community Dynamics in Evolutionary Networks. In *The Proceedings of the 10th IEEE International Conference on Data Mining Workshops (ICDMW' 10)*, pages 318–327. IEEE Computer Society, 2010. (Cited on p. 111)

[46] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985. (Cited on p. 179)

[47] Markus Chimani, Maria Kandyba, Ivana Ljubic, and Petra Mutzel. Obtaining optimal k-cardinality trees fast. *ACM Journal of Experimental Algorithmics*, 14, 2009. (Cited on p. 21)

[48] Jon M Conrad, Carla P Gomes, Willem-Jan van Hoeve, Ashish Sabharwal, and Jordan F Suter. Wildlife corridors as a connected subgraph problem. *Journal of Environmental Economics and Management*, 63(1):1–18, 2012. (Cited on pp. 20, 21, 80)

[49] Alessio Conte, Donatella Firmani, Caterina Mordente, Maurizio Patrignani, and Riccardo Torlone. Fast Enumeration of Large k-Plexes. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*, pages 115–124. ACM, 2017. (Cited on p. 111)

[50] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. D2K: Scalable Community Detection in Massive Networks via Small-Diameter k-Plexes. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '18)*, pages 1272–1281. ACM, 2018. (Cited on p. 111)

[51] Emilio Coppa, Irene Finocchi, and Renan Leon Garcia. Counting cliques in parallel without a cluster: Engineering a fork/join algorithm for shared-memory platforms. *Information Sciences*, 496:553–571, 2019. (Cited on p. 110)

[52] Jean-François Couturier and Dieter Kratsch. Bicolored independent sets and bicliques. *Information Processing Letters*, 112(8-9):329–334, 2012. (Cited on pp. 129, 130)

[53] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. (Cited on pp. 31, 32, 33, 35, 112, 202)

[54] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. (Cited on p. 142)

[55] Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed-parameter tractable. *SIAM Journal on Computing*, 48(2):417–450, 2019. (Cited on pp. 201, 250)

[56] Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '12)*, pages 68–81. SIAM, 2012. (Cited on p. 35)

[57] Matjaz Depolli, Janez Konc, Kati Rozman, Roman Trobec, and Dusanka Janezic. Exact Parallel Maximum Clique Algorithm for General and Protein Graphs. *Journal of Chemical Information and Modeling*, 53(9):2217–2228, 2013. (Cited on p. 21)

[58] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. (Cited on p. 27)

[59] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense implicit communities in the web graph. *ACM Transactions on the Web*, 3(2):7:1–7:36, 2009. (Cited on p. 199)

[60] Rodney G. Downey, Vladimir Estivill-Castro, Michael R. Fellows, Elena Prieto-Rodriguez, and Frances A. Rosamond. Cutting up is hard to do: the parameterized complexity of k-cut and related problems. *Electronic Notes in Theoretical Computer Science*, 78:209–222, 2003. (Cited on pp. 79, 201)

[61] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995. (Cited on p. 32)

[62] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theoretical Computer Science*, 141(1&2):109–131, 1995. (Cited on pp. 32, 200, 250)

[63] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. (Cited on pp. 31, 32, 112, 142, 202)

[64] Pavel Dvorák, Andreas Emil Feldmann, Ashutosh Rai, and Pawel Rzazewski. Parameterized inapproximability of independent set in H-free graphs. In *Proceedings of the 46th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '20)*, volume 12301 of *Lecture Notes in Computer Science*, pages 40–53. Springer, 2020. (Cited on p. 238)

[65] Nils Jakob Eckstein, Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer. Destroying multicolored paths and cycles in edge-colored graphs. *CoRR*, abs/2104.03138, 2021. (Cited on p. 7)

[66] Matthias Ehrgott and Horst W. Hamacher. *Fixed Cardinality Combinatorial Optimization Problems - A Survey*. Number 56 in Report in Wirtschaftsmathematik (WIMA Report). 1999. (Cited on pp. 80, 199)

[67] Khaled M. Elbassioni. A polynomial delay algorithm for generating connected induced subgraphs of a given cardinality. *Journal of Graph Algorithms and Applications*, 19(1):273–280, 2015. (Cited on pp. 20, 38, 39, 40, 41, 42, 56, 58, 59, 75, 253)

[68] Shady Elbassuoni and Roi Blanco. Keyword search over RDF graphs. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management (CIKM '11)*, pages 237–242. ACM, 2011. (Cited on pp. 37, 64)

[69] David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters*, 51(4):207–211, 1994. (Cited on p. 126)

[70] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 18, 2013. (Cited on pp. 19, 112, 254)

[71] David Eppstein and Emma S. Spiro. The h-index of a graph and its application to dynamic subgraph statistics. *Journal of Graph Algorithms and Applications*, 16(2):543–567, 2012. (Cited on p. 28)

[72] Zeynep Ertem, Alexander Veremyev, and Sergiy Butenko. Detecting large cohesive subgroups with high clustering coefficients in social networks. *Social Networks*, 46:1–10, 2016. (Cited on p. 138)

[73] M. G. Everett and S. P. Borgatti. The centrality of groups and classes. *The Journal of Mathematical Sociology*, 23(3):181–201, 1999. (Cited on pp. 25, 199)

[74] Uriel Feige, Guy Kortsarz, and David Peleg. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001. (Cited on pp. 79, 83, 200, 256)

[75] Uriel Feige, Robert Krauthgamer, and Kobbi Nissim. On cutting a few vertices from a graph. *Discrete Applied Mathematics*, 127(3):643–649, 2003. (Cited on p. 201)

[76] Uriel Feige and Michael Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *Journal of Algorithms*, 41(2):174–211, 2001. (Cited on p. 201)

[77] Uriel Feige and Michael Seltser. On the densest k-subgraph problem. Technical report, Weizmann Institute of Science. Department of Applied Mathematics and Computer Science, 1997. (Cited on pp. 79, 83, 218, 219)

[78] Qilong Feng, Shaohua Li, Zeyang Zhou, and Jianxin Wang. Parameterized algorithms for edge biclique and related problems. *Theoretical Computer Science*, 734:105–118, 2018. (Cited on p. 126)

[79] Rui Ferreira. *Efficiently Listing Combinatorial Patterns in Graphs*. PhD thesis, Dipartimento di Informatica, Università degli Studi di Pisa, 2013. Available at https://arxiv.org/abs/1308.6635. (Cited on pp. 39, 77)

[80] Matteo Fischetti, Horst W. Hamacher, Kurt Jørnsten, and Francesco Maffioli. Weighted k-cardinality trees: Complexity and polyhedral structure. *Networks*, 24(1):11–21, 1994. (Cited on pp. 19, 80)

[81] Gary William Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of Web communities. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*, pages 150–160. ACM, 2000. (Cited on pp. 19, 21)

[82] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. (Cited on p. 31)

[83] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010. (Cited on pp. 19, 23)

[84] Jacob Fox, Tim Roughgarden, C. Seshadhri, Fan Wei, and Nicole Wein. Finding cliques in social networks: A new distribution-free model. *SIAM Journal on Computing*, 49(2):448–464, 2020. (Cited on pp. 24, 28, 113, 114, 116, 122, 133, 135, 202, 249, 254)

[85] Rajiv Gandhi, Samir Khuller, and Aravind Srinivasan. Approximation algorithms for partial covering problems. *Journal of Algorithms*, 53(1):55–84, 2004. (Cited on pp. 199, 200)

[86] Rajiv Gandhi and Guy Kortsarz. On set expansion problems and the small set expansion conjecture. *Discrete Applied Mathematics*, 194:93–101, 2015. (Cited on p. 200)

[87] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. (Cited on pp. 19, 21, 29, 79, 130, 200, 201)

[88] Jaroslav Garvardt, Christian Komusiewicz, and Frank Sommer. The Parameterized Complexity of s-Club with Triangle and Seed Constraints. In *Proceedings of the 33rd International Workshop on Combinatorial Algorithms (IWOCA '22)*, volume 13270 of *Lecture Notes in Computer Science*, pages 313–326. Springer, 2022. (Cited on pp. 9, 177)

[89] Jaroslav Garvardt, Christian Komusiewicz, and Frank Sommer. The parameterized complexity of s-club with triangle and seed constraints. *CoRR*, abs/2201.05654, 2022. URL: `https://arxiv.org/abs/2201.05654`. (Cited on p. 9)

[90] Serge Gaspers, Dieter Kratsch, and Mathieu Liedloff. On independent sets and bicliques in graphs. *Algorithmica*, 62(3-4):637–658, 2012. (Cited on pp. 128, 130)

[91] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB '05)*, pages 721–732. ACM, 2005. (Cited on p. 199)

[92] O Goldschmidt, DS Hochbaum, and G Yu. A simulated annealing heuristic for the semiconductor components assembly. Technical report, IE & OR Department UC Berkeley, CA, 1994. (Cited on p. 199)

[93] Petr A. Golovach, Pinar Heggernes, Dieter Kratsch, and Arash Rafiey. Finding clubs in graph classes. *Discrete Applied Mathematics*, 174:57–65, 2014. (Cited on p. 141)

[94] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. On the acyclic subgraph polytope. *Mathematical Programming*, 33(1):28–42, 1985. (Cited on p. 79)

[95] Niels Grüttemeier, Philipp Heinrich Keßler, Christian Komusiewicz, and Frank Sommer. Efficient branch-and-bound algorithms for finding triangle-constrained 2-clubs. *CoRR*, abs/2211.01701, 2022. URL: `https://doi.org/10.48550/arXiv.2211.01701`. (Cited on p. 9)

[96] Niels Grüttemeier, Christian Komusiewicz, Nils Morawietz, and Frank Sommer. String factorizations under various collision constraints. In *Proceedings of the Thirty-First Annual Symposium on Combinatorial Pattern Matching (CPM'20)*, volume 161 of *LIPIcs*, pages 17:1–17:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited on p. 8)

[97] Niels Grüttemeier, Christian Komusiewicz, Nils Morawietz, and Frank Sommer. Preventing small (s, t)-cuts by protecting edges. In *Proceedings of the 47th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '21)*, volume 12911 of *Lecture Notes in Computer Science*, pages 143–155. Springer, 2021. (Cited on p. 8)

[98] Niels Grüttemeier, Christian Komusiewicz, Jannik Schestag, and Frank Sommer. Destroying bicolored P_3s by deleting few edges. In *Proceedings of the 15th Conference on Computability in Europe (CiE '19)*, volume 11558 of *Lecture Notes in Computer Science*, pages 193–204. Springer, 2019. (Cited on p. 7)

[99] Niels Grüttemeier, Christian Komusiewicz, Jannik Schestag, and Frank Sommer. Destroying bicolored P_3s by deleting few edges. *Discrete Mathematics & Theoretical Computer Science*, 23(1), 2021. (Cited on p. 7)

[100] Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM Journal on Discrete Mathematics*, 24(4):1662–1683, 2010. (Cited on p. 255)

[101] Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of vertex cover variants. *Theory of Computing Systems*, 41(3):501–520, 2007. (Cited on pp. 25, 200, 202)

[102] Anupam Gupta, Euiwoong Lee, and Jason Li. An FPT algorithm beating 2-approximation for k-cut. In *Proceedings of the Twenty-Ninth Annual*

ACM-SIAM Symposium on Discrete Algorithms (SODA '18), pages 2821–2837. SIAM, 2018. (Cited on p. 200)

[103] Pietro Hiram Guzzi and Swarup Roy. *Biological Network Analysis: Trends, Approaches, Graph Theory, and Algorithms*. Elsevier, 2020. (Cited on p. 19)

[104] Eran Halperin and Aravind Srinivasan. Improved approximation algorithms for the partial vertex cover problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX '02)*, volume 2462 of *Lecture Notes in Computer Science*, pages 161–174. Springer, 2002. (Cited on p. 200)

[105] H. W. Hamacher and K. Joernstern. Optimal relinquishments. Technical report, Norwegian School of Economics and Business Administration, 1992. (Cited on pp. 20, 21, 80)

[106] Satoshi Hara, Takayuki Katsuki, Hiroki Yanagisawa, Takafumi Ono, Ryo Okamoto, and Shigeki Takeuchi. Consistent and efficient nonparametric different-feature selection. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS '17)*, volume 54 of *Proceedings of Machine Learning Research*, pages 130–138. PMLR, 2017. (Cited on p. 79)

[107] Satoshi Hara, Tetsuro Morimura, Toshihiro Takahashi, Hiroki Yanagisawa, and Taiji Suzuki. A consistent method for graph based anomaly localization. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics (AISTATS '15)*, volume 38 of *Proceedings of Machine Learning Research*, pages 333–341. PMLR, 2015. (Cited on p. 79)

[108] Kazuya Haraguchi and Hiroshi Nagamochi. Enumeration of Support-Closed Subsets in Confluent Systems. *Algorithmica*, pages 1–37, 2022. (Cited on pp. 40, 60, 62)

[109] Sepp Hartung, Christian Komusiewicz, and André Nichterlein. Parameterized algorithmics and computational experiments for finding 2-clubs. *Journal of Graph Algorithms and Applications*, 19(1):155–190, 2015. (Cited on pp. 19, 24, 137, 141, 171, 172, 176, 177, 181, 184, 256)

[110] Sepp Hartung, Christian Komusiewicz, André Nichterlein, and Ondrej Suchý. On structural parameterizations for the 2-club problem. *Discrete Applied Mathematics*, 185:79–92, 2015. (Cited on pp. 24, 113, 122, 141, 171, 254, 255)

[111] Danny Hermelin and George Manoussakis. Efficient enumeration of maximal induced bicliques. *Discrete Applied Mathematics*, 303:253–261, 2021. (Cited on pp. 113, 130, 253, 254)

[112] Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '12)*, pages 104–113. SIAM, 2012. (Cited on pp. 34, 35)

[113] Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the Bron-Kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining*, 7(1):35:1–35:16, 2017. (Cited on p. 253)

[114] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(04):439–562, 2006. (Cited on p. 222)

[115] Wei-Qiang Huang, Xin-Tian Zhuang, and Shuang Yao. A network analysis of the Chinese stock market. *Physica A: Statistical Mechanics and its Applications*, 388(14):2956–2964, 2009. (Cited on p. 21)

[116] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION '11)*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, 2011. (Cited on p. 109)

[117] Tai Huynh, Ivan Zelinka, Xuan Hau Pham, and Hien D. Nguyen. Some measures to detect the influencer on social network based on information propagation. In *Proceedings of the 9th International Conference on Web Intelligence, Mining and Semantics (WIMS '19)*, pages 18:1–18:6. ACM, 2019. (Cited on p. 19)

[118] Hiro Ito and Kazuo Iwama. Enumeration of isolated cliques and pseudo-cliques. *ACM Transactions on Algorithms*, 5(4):40:1–40:21, 2009. (Cited on p. 255)

[119] Yoichi Iwata, Keigo Oka, and Yuichi Yoshida. Linear-time FPT algorithms via network flow. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '14)*, pages 1749–1761. SIAM, 2014. (Cited on p. 257)

[120] D. Jallo, V. Budai, V. Boginski, B. Goldengorin, and P.M. Pardalos. Network-based representation of stock market dynamics: An application to american and swedish stock markets. In *Springer Proceedings in Mathematics & Statistics*, volume 32, pages 91 – 108. Operations Management & Operations Research, 2013. (Cited on pp. 19, 23)

[121] Mark Jerrum and Kitty Meeks. The parameterised complexity of counting connected subgraphs and graph motifs. *Journal of Computer and System Sciences*, 81(4):702–716, 2015. (Cited on p. 77)

[122] Stasys Jukna. *Extremal Combinatorics - With Applications in Computer Science*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2011. (Cited on p. 29)

[123] Björn H Junker and Falk Schreiber. *Analysis of Biological Networks*. John Wiley & Sons, 2011. (Cited on p. 19)

[124] Rushed Kanawati. Seed-centric approaches for community detection in complex networks. In *Proceedings of the 6th International Conference on Social Computing and Social Media (SCSM '14)*, volume 8531 of *Lecture Notes in Computer Science*, pages 197–208. Springer, 2014. (Cited on pp. 24, 141, 255)

[125] Lawqueen Kanesh, Jayakrishnan Madathil, Sanjukta Roy, Abhishek Sahu, and Saket Saurabh. Further Exploiting c-Closure for FPT Algorithms and Kernels for Domination Problems. In *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS '22)*, volume 219 of *LIPIcs*, pages 39:1–39:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on pp. 110, 114, 249, 254)

[126] Richard M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. (Cited on pp. 19, 21, 200, 201)

[127] Zahra Razaghi Moghadam Kashani, Hayedeh Ahrabian, Elahe Elahi, Abbas Nowzari-Dalini, Elnaz Saberi Ansari, Sahar Asadi, Shahin Mohammadi, Falk Schreiber, and Ali Masoudi-Nejad. Kavosh: a new algorithm for finding network motifs. *BMC Bioinformatics*, 10:318, 2009. (Cited on pp. 20, 37, 39, 41, 71, 76, 253, 287, 289)

[128] Maximilian Katzmann and Christian Komusiewicz. Systematic Exploration of Larger Local Search Neighborhoods for the Minimum Vertex Cover Problem. In *Proceedings of the Thirty-First Conference on Artificial Intelligence (AAAI '17)*, pages 846–852. AAAI Press, 2017. (Cited on p. 80)

[129] Philipp Heinrich Keßler. Algorithm engineering for the triangle-2-club problem. Bachelor's thesis, Philipps-Universität Marburg, 2020. URL: `https://www.uni-marburg.de/de/fb12/arbeitsgruppen/algorith/paper/philipp_kessler_bachelor-thesis.pdf`. (Cited on p. 10)

[130] Subhash Khot. Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM Journal on Computing*, 36(4):1025–1071, 2006. (Cited on p. 200)

[131] Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002. (Cited on pp. 24, 112)

[132] Narsis A Kiani, David Gomez-Cabrero, and Ginestra Bianconi. *Networks of Networks in Biology: Concepts, Tools and Applications*. Cambridge University Press, 2021. (Cited on p. 19)

[133] Joachim Kneis, Alexander Langer, and Peter Rossmanith. Improved upper bounds for partial vertex cover. In *Proceedings of the 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '08)*, volume 5344 of *Lecture Notes in Computer Science*, pages 240–251, 2008. (Cited on pp. 25, 200, 250, 256)

[134] Joachim Kneis, Daniel Mölle, and Peter Rossmanith. Partial vs. complete domination: t-dominating set. In *Proceedings of the 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '07)*, volume 4362 of *Lecture Notes in Computer Science*, pages 367–376. Springer, 2007. (Cited on p. 25)

[135] Tomohiro Koana, Christian Komusiewicz, André Nichterlein, and Frank Sommer. Covering many (or few) Edges with k vertices in sparse graphs. In *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS '22)*, volume 219 of *LIPIcs*, pages 42:1–42:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on p. 10)

[136] Tomohiro Koana, Christian Komusiewicz, André Nichterlein, and Frank Sommer. Covering many (or few) edges with k vertices in sparse graphs. *CoRR*, abs/2201.05465, 2022. URL: `https://arxiv.org/abs/2201.05465`, `arXiv:2201.05465`. (Cited on p. 10)

[137] Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Computing dense and sparse subgraphs of weakly closed graphs. In *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC '20)*, volume 181 of *LIPIcs*, pages 20:1–20:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited on pp. 9, 114, 129, 134, 203, 233, 254)

[138] Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Computing dense and sparse subgraphs of weakly closed graphs. *CoRR*, abs/2007.05630, 2020. URL: `https://arxiv.org/abs/2007.05630`. (Cited on p. 9)

[139] Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Exploiting c-closure in kernelization algorithms for graph problems. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA '20)*, volume 173 of *LIPIcs*, pages 65:1–65:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited on pp. 7, 114, 222)

[140] Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Essentially tight kernels for (weakly) closed graphs. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC' 21)*, volume 212 of *LIPIcs*, pages 35:1–35:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on pp. 7, 110, 254)

[141] Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Exploiting c-closure in kernelization algorithms for graph problems. *SIAM Journal on Discrete Mathematics*, 36(4):2798–2821, 2022. (Cited on p. 7)

[142] Tomohiro Koana and André Nichterlein. Detecting and enumerating small induced subgraphs in c-closed graphs. *Discrete Applied Mathematics*, 302:198–207, 2021. (Cited on pp. 114, 135)

[143] Christian Komusiewicz. Multivariate algorithmics for finding cohesive subnetworks. *Algorithms*, 9(1):21, 2016. (Cited on pp. 19, 22, 24, 112, 133, 254)

[144] Christian Komusiewicz, Falk Hüffner, Hannes Moser, and Rolf Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theoretical Computer Science*, 410(38-40):3640–3654, 2009. (Cited on pp. 112, 255)

[145] Christian Komusiewicz, André Nichterlein, Rolf Niedermeier, and Marten Picker. Exact algorithms for finding well-connected 2-clubs in sparse real-world graphs: Theory and experiments. *European Journal of Operational Research*, 275(3):846–864, 2019. (Cited on pp. 19, 24, 137, 141, 173, 176, 177, 181, 184)

[146] Christian Komusiewicz and Frank Sommer. Enumerating connected induced subgraphs: Improved delay and experimental comparison. In *Proceedings of the 45th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '19)*, volume 11376 of *Lecture Notes in Computer Science*, pages 272–284. Springer, 2019. (Cited on p. 8)

[147] Christian Komusiewicz and Frank Sommer. FixCon: A generic solver for fixed-cardinality subgraph problems. In *Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX '20)*, pages 12–26. SIAM, 2020. (Cited on p. 9)

[148] Christian Komusiewicz and Frank Sommer. Enumerating connected induced subgraphs: Improved delay and experimental comparison. *Discrete Applied Mathematics*, 303:262–282, 2021. (Cited on p. 8)

[149] Christian Komusiewicz and Manuel Sorge. Finding dense subgraphs of sparse graphs. In *Proceedings of the 7th International Symposium on Parameterized and Exact Computation (IPEC '12)*, volume 7535 of *Lecture Notes in Computer Science*, pages 242–251. Springer, 2012. (Cited on pp. 39, 51)

[150] Christian Komusiewicz and Manuel Sorge. An algorithmic framework for fixed-cardinality optimization in sparse graphs applied to dense subgraph problems. *Discrete Applied Mathematics*, 193:145–161, 2015. (Cited on pp. 20, 21, 37, 38, 39, 41, 77, 79, 81, 82, 200, 218, 247, 285, 287)

[151] Christian Komusiewicz, Manuel Sorge, and Kolja Stahl. Finding connected subgraphs of fixed minimum density: Implementation and experiments. In *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA '15)*, volume 9125 of *Lecture Notes in Computer Science*, pages 82–93. Springer, 2015. (Cited on pp. 21, 37, 81, 101)

[152] Donald L. Kreher and Douglas R. Stinson. *Combinatorial Algorithms: Generation, Enumeration, and Search (Discrete Mathematics and Its Applications)*. CRC Press, 1998. (Cited on p. 71)

[153] Jérôme Kunegis. KONECT: the Koblenz network collection. In *Proceedings of the 22nd International World Wide Web Conference (WWW '13)*, pages 1343–1350. International World Wide Web Conferences Steering Committee / ACM, 2013. (Cited on pp. 68, 86, 191)

[154] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '09)*, pages 467–476. ACM, 2009. (Cited on p. 19)

[155] Thomas Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Applicable Theory in Computer Science. Teubner, 1990. (Cited on p. 199)

[156] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. (Cited on pp. 22, 23)

[157] Bingkai Lin. The parameterized complexity of the k-biclique problem. *Journal of the ACM*, 65(5):34:1–34:23, 2018. (Cited on p. 112)

[158] Hong Liu, Peng Zhang, and Daming Zhu. On editing graphs into 2-club clusters. In *Proceedings of the Joint International Conference in Frontiers in Algorithmics and Algorithmic Aspects in Information and Management (FAW-AAIM '12)*, volume 7285 of *Lecture Notes in Computer Science*, pages 235–246. Springer, 2012. (Cited on p. 255)

[159] William Lochet, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Exploiting dense structures in parameterized complexity. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS '21)*, volume 187 of *LIPIcs*, pages 50:1–50:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on p. 250)

[160] Daniel Lokshtanov and Vaishali Surianarayanan. Dominating set in weakly closed graphs is fixed parameter tractable. In *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '21)*, volume 213 of *LIPIcs*, pages 29:1–29:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on p. 114)

[161] R Duncan Luce. Connectivity and generalized cliques in sociometric group structure. *Psychometrika*, 15(2):169–190, 1950. (Cited on p. 133)

[162] Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999. (Cited on p. 257)

[163] Zoran Maksimović. A new mixed integer linear programming formulation for the maximum degree bounded connected subgraph problem. *Publications de l'Institut Mathematique*, 99(113):99–108, 2016. (Cited on p. 108)

[164] Pasin Manurangsi. A note on max k-vertex cover: Faster FPT-AS, smaller approximate kernel and improved approximation. In *Proceedings of the 2nd Symposium on Simplicity in Algorithms (SOSA '19)*, volume 69 of *OASICS*, pages 15:1–15:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited on p. 200)

[165] Andrea Marino. *Analysis and Enumeration: Algorithms for Biological Graphs*, volume 6. Atlantis Press, 2015. (Cited on p. 36)

[166] Hideo Matsuda, T. Ishihara, and Akihiro Hashimoto. Classifying Molecular Sequences Using a Linkage Graph With Their Pairwise Similarities. *Theoretical Computer Science*, 210(2):305–325, 1999. (Cited on p. 133)

[167] Sean Maxwell, Mark R. Chance, and Mehmet Koyutürk. Efficiently enumerating all connected induced subgraphs of a large molecular network. In *Proceedings of the First International Conference on Algorithms for Computational Biology (AlCoB '14)*, volume 8542 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2014. (Cited on pp. 39, 41, 75, 77, 81, 82, 108, 289, 292)

[168] Benjamin McClosky. Clique relaxations. *Wiley Encyclopedia of Operations Research and Management Science*, 2010. (Cited on pp. 22, 111)

[169] Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016. (Cited on p. 253)

[170] Robert J Mokken et al. Cliques, clubs and clans. *Quality & Quantity*, 13(2):161–173, 1979. (Cited on p. 23)

[171] Hendrik Molter, Rolf Niedermeier, and Malte Renken. Enumerating isolated cliques in temporal networks. In *Proceedings of the Eighth International Conference on Complex Networks and Their Applications (COMPLEX NETWORKS '19)*, volume 882 of *Studies in Computational Intelligence*, pages 519–531. Springer, 2019. (Cited on p. 255)

[172] John W Moon and Leo Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, 1965. (Cited on pp. 111, 125)

[173] Nils Morawietz, Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer. Colored cut games. In *Proceedings of the 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '20)*, volume 182 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited on p. 7)

[174] Nils Morawietz, Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer. Refined parameterizations for computing colored cuts in edge-colored graphs. In *Proceedings of the 46th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '20)*, volume 12011 of *Lecture Notes in Computer Science*, pages 248–259. Springer, 2020. (Cited on p. 8)

[175] Nils Morawietz, Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer. Colored cut games. *Theoretical Computer Science*, 936:13–32, 2022. (Cited on p. 7)

[176] Nils Morawietz, Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer. Refined parameterizations for computing colored cuts in edge-colored graphs. *Theory of Computing Systems*, 66(5):1019–1045, 2022. (Cited on p. 8)

[177] N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. LP can be a cure for parameterized problems. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS '12)*, volume 14 of *LIPIcs*, pages 338–349. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. (Cited on p. 257)

[178] Jesper Nederlof, Johan M. M. van Rooij, and Thomas C. van Dijk. Inclusion/exclusion meets measure and conquer. *Algorithmica*, 69(3):685–740, 2014. (Cited on p. 25)

[179] Mark E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003. (Cited on p. 137)

[180] Mark E. J. Newman. *Networks: An Introduction*. Oxford University Press, 2010. (Cited on p. 251)

[181] Lam B. Q. Nguyen, Loan T. T. Nguyen, Ivan Zelinka, Václav Snásel, Hung Son Nguyen, and Bay Vo. A method for closed frequent subgraph mining in a single large graph. *IEEE Access*, 9:165719–165733, 2021. (Cited on p. 38)

[182] Lam BQ Nguyen, Ivan Zelinka, and Quoc Bao Diep. CCGraMi: An effective method for mining frequent subgraphs in a single large graph. *MENDEL Soft Computing Journal*, 27(2):90–99, 2021. (Cited on p. 38)

[183] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and Its Applications. OUP Oxford, 2006. (Cited on p. 31)

[184] Mark Ortmann and Ulrik Brandes. Efficient orbit-aware triad and quad census in directed and undirected graphs. *Applied Network Science*, 2:13, 2017. (Cited on p. 77)

[185] Foad Mahdavi Pajouh, Balabhaskar Balasundaram, and Illya V. Hicks. On the 2-club polytope of graphs. *Operations Research*, 64(6):1466–1481, 2016. (Cited on pp. 19, 24, 176)

[186] Foad Mahdavi Pajouh, Zhuqi Miao, and Balabhaskar Balasundaram. A branch-and-bound approach for maximum quasi-cliques. *Annals of Operations Research*, 216(1):145–161, 2014. (Cited on p. 101)

[187] Fahad Panolan and Hannane Yaghoubizade. Partial vertex cover on graphs of bounded degeneracy. *CoRR*, abs/2201.03876, 2022. URL: `https://arxiv.org/abs/2201.03876`. (Cited on pp. 200, 203, 237, 249, 256)

[188] Christos H. Papadimitriou. *Computational complexity*. Academic Internet Public, 2007. (Cited on p. 29)

[189] Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. Motifs in Temporal Networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17)*, pages 601–610. ACM, 2017. (Cited on p. 253)

[190] Srinivas Pasupuleti. Detection of protein complexes in protein interaction networks using n-clubs. In *Proceedings of the 6th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBIO' 08)*, volume 4973 of *Lecture Notes in Computer Science*, pages 153–164. Springer, 2008. (Cited on p. 23)

[191] Jeffrey Pattillo, Alexander Veremyev, Sergiy Butenko, and Vladimir Boginski. On the maximum quasi-clique problem. *Discrete Applied Mathematics*, 161(1-2):244–257, 2013. (Cited on pp. 22, 111)

[192] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. Clique relaxation models in social network analysis. In *Handbook of Optimization in Complex Networks*, pages 143–162. Springer, 2012. (Cited on p. 22)

[193] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013. (Cited on pp. 22, 24, 111, 137, 141, 173)

[194] René Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003. (Cited on p. 128)

[195] Jian Pei, Daxin Jiang, and Aidong Zhang. On mining cross-graph quasi-cliques. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '05)*, pages 228–238. ACM, 2005. (Cited on p. 133)

[196] Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. ESCAPE: efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*, pages 1431–1440. ACM, 2017. (Cited on p. 77)

[197] Frederico Paiva Quintão, Alexandre Salles da Cunha, Geraldo Robson Mateus, and Abilio Lucena. The k-cardinality tree problem: Reformulations and lagrangian relaxation. *Discrete Applied Mathematics*, 158(12):1305–1314, 2010. (Cited on p. 21)

[198] Venkatesh Raman and Saket Saurabh. Short cycles make W-hard problems hard: FPT algorithms for W-hard problems in graphs with no short cycles. *Algorithmica*, 52(2):203–225, 2008. (Cited on pp. 24, 112, 116, 200, 203, 233, 234)

[199] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. (Cited on p. 249)

[200] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI '15)*, pages 4292–4293. AAAI Press, 2015. URL: `http://networkrepository.com`. (Cited on pp. 68, 86, 191)

[201] Suparna Saha, Saurav Mallik, and Sanghamitra Bandyopadhyay. DeMoS: Dense module based gene signature detection through quasi-clique: An application to cervical cancer prognosis, 2020. URL: `https://doi.org/10.21203/rs.3.rs-17212/v1`. (Cited on p. 23)

[202] Saeed Salem, Mohammed Alokshiya, and Mohammad Al Hasan. RASMA: a reverse search algorithm for mining maximal frequent subgraphs. *BioData Mining*, 14(1):19, 2021. (Cited on pp. 38, 40, 41, 63, 76)

[203] Hosseinali Salemi and Austin Buchanan. Parsimonious formulations for low-diameter clusters. *Mathematical Programming Computation*, 12(3):493–528, 2020. (Cited on pp. 19, 24, 176)

[204] Saket Saurabh and Meirav Zehavi. $(k, n-k)$-MAX-CUT: An $\mathcal{O}^*(2^p)$-time algorithm and a polynomial kernel. *Algorithmica*, 80(12):3844–3860, 2018. (Cited on pp. 25, 201, 250, 256)

[205] Saket Saurabh and Meirav Zehavi. Parameterized complexity of multi-node hubs. *Journal of Computer and System Sciences*, 131:64–85, 2023. (Cited on pp. 19, 20, 21, 110, 250, 256)

[206] Alexander Schäfer, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Parameterized computational complexity of finding small-diameter subgraphs. *Optimization Letters*, 6(5):883–891, 2012. (Cited on pp. 112, 141, 178)

[207] Jannik Schestag, Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer. On critical node problems with vulnerable vertices. In *Proceedings of the 33rd International Workshop on Combinatorial Algorithms (IWOCA '22)*, volume 13270 of *Lecture Notes in Computer Science*, pages 494–508. Springer, 2022. (Cited on p. 8)

[208] Stephen B Seidman and Brian L Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6(1):139–154, 1978. (Cited on p. 22)

[209] Hadas Shachnai and Meirav Zehavi. Parameterized algorithms for graph partitioning problems. *Theory of Computing Systems*, 61(3):721–738, 2017. (Cited on pp. 202, 235, 250, 256)

[210] Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004. (Cited on p. 255)

[211] Roded Sharan, Igor Ulitsky, and Ron Shamir. Network-based prediction of protein function. *Molecular Systems Biology*, 3(1):88, 2007. (Cited on pp. 19, 21)

[212] Raymond M Smullyan. *First-Order Logic*. Courier Corporation, 1995. (Cited on p. 110)

[213] Yann Strozecki. *Enumeration Complexity and Matroid Decomposition*. PhD thesis, Université Paris Diderot - Paris 7, 2010. (Cited on p. 36)

[214] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. (Cited on p. 65)

[215] Etsuji Tomita, Tatsuya Akutsu, and Tsutomu Matsunaga. Efficient Algorithms for Finding Maximum and Maximal Cliques: Effective Tools for Bioinformatics. In *Biomedical Engineering, Trends in Electronics*, chapter 32. IntechOpen, 2011. (Cited on pp. 19, 21)

[216] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006. (Cited on p. 111)

[217] Charalampos E. Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*, pages 1122–1132. ACM, 2015. (Cited on p. 79)

[218] Takeaki Uno. Two general methods to reduce delay and change of enumeration algorithms. Technical report, National Institute of Informatics, 2003. (Cited on pp. 60, 64, 66)

[219] Takeaki Uno. Constant time enumeration by amortization. In *Proceedings of the 14th International Symposium on Algorithms and Data Structures (WADS '15)*, volume 9214 of *Lecture Notes in Computer Science*, pages 593–605. Springer, 2015. (Cited on p. 40)

[220] René van Bevern, Sepp Hartung, Frank Kammer, Rolf Niedermeier, and Mathias Weller. Linear-time computation of a linear problem kernel for dominating set on planar graphs. In *Proceedings of the 6th International Symposium on Parameterized and Exact Computation (IPEC '11)*, volume 7112 of *Lecture Notes in Computer Science*, pages 194–206. Springer, 2011. (Cited on p. 257)

[221] René van Bevern, Hannes Moser, and Rolf Niedermeier. Approximation and tidying - a problem kernel for s-plex cluster vertex deletion. *Algorithmica*, 62(3-4):930–950, 2012. (Cited on p. 255)

[222] Alexander Veremyev and Vladimir Boginski. Identifying large robust network clusters via new compact formulations of maximum k-club problems. *European Journal of Operational Research*, 218(2):316–326, 2012. (Cited on pp. 24, 137, 141, 173)

[223] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994. (Cited on pp. 19, 21, 253)

[224] Rémi Watrigant, Marin Bougeret, and Rodolphe Giroudeau. Approximating the sparsest k-subgraph in chordal graphs. *Theory of Computing Systems*, 58(1):111–132, 2016. (Cited on pp. 79, 200)

[225] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *Nature*, 393(6684):440–442, 1998. (Cited on p. 137)

[226] Dominic Welsh and DJA Welsh. *Complexity: Knots, Colourings and Countings*, volume 186. Cambridge university press, 1993. (Cited on p. 199)

[227] Sebastian Wernicke. Efficient detection of network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):347–359, 2006. (Cited on pp. 20, 37, 39, 43, 44, 46, 50, 253)

[228] Sebastian Wernicke and Florian Rasche. FANMOD: a tool for fast network motif detection. *Bioinformatics*, 22(9):1152–1153, 2006. (Cited on p. 53)

[229] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, 2000. (Cited on p. 27)

[230] Joyce Jiyoung Whang, David F. Gleich, and Inderjit S. Dhillon. Overlapping community detection using seed set expansion. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management (CIKM '13)*, pages 2099–2108. ACM, 2013. (Cited on pp. 24, 141, 255)

[231] Oleksandra Yezerska, Foad Mahdavi Pajouh, and Sergiy Butenko. On biconnected and fragile subgraphs of low diameter. *European Journal of Operational Research*, 263(2):390–400, 2017. (Cited on pp. 141, 173)

[232] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006. (Cited on pp. 19, 23)

[233] Bing Zhang, Byung-Hoon Park, Tatiana V. Karpinets, and Nagiza F. Samatova. From pull-down data to protein interaction networks and complexes with biological relevance. *Bioinformatics*, 24(7):979–986, 2008. (Cited on p. 111)

[234] Peng Zhang. Unbalanced graph cuts with minimum capacity. *Frontiers in Computer Science*, 8(4):676–683, 2014. (Cited on p. 201)

[235] Yupeng Zhou, Changze Qiu, Yiyuan Wang, Mingjie Fan, and Minghao Yin. An improved memetic algorithm for the partial vertex cover problem. *IEEE Access*, 7:17389–17402, 2019. (Cited on p. 200)

[236] David Zuckerman. Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*, pages 681–690. ACM, 2006. (Cited on p. 254)

Appendix A

Further Algorithms Used in the Experiments for EnuCon

In this chapter, we the describe the other algorithms that we include in our experimental comparison for `EnuCon` (see Chapter 3). For `Exgen` and `Kavosh`, we provide a pruning rule that will be useful in the case where k is big. For `Kavosh` and `BDDE`, we provide the the first running time bounds based on Lemma 3.1. Afterwards, we describe the `RSSP` algorithm [4].

A.1 Exgen

The next algorithm, referred to as `Exgen` in the following, is a variant of `Pivot`. The pseudocode of `Exgen` (including the new pruning rule (Lines 13 and 15)) is shown in Algorithm A.1.

The `Exgen` algorithm was described by Komusiewicz and Sorge [150] since, compared to `Pivot`, it was easier to bound the number of recursive calls by the number of E-CISE solutions. The sets P, S, and F are defined as in Section 3.3, that is, P contains the vertices of the subgraph set whose neighbors may still be added, S contains the other vertices of the subgraph set, and F contains the vertices which may not be added to the subgraph anymore. In each recursive call, we choose one pivot vertex p from the set P and determine the set X of its neighbors which are not in $P \cup S \cup F$. Next, we move the pivot vertex p from set P to set S, since in the recursive calls of `Exgen`, no further neighbors of p may be added to the subgraph. Afterwards, for each subset M of X which has at most $k - |P \cup S|$ vertices, we call `Exgen` recursively with M added to the subgraph set P (the size bound comes from the fact that we search for subgraphs of order k only). Note that $M = \emptyset$ is a valid

Algorithm A.1: The `Exgen` algorithm; the initial call is `Exgen`$(\{v\}, \emptyset, \emptyset)$.

```
1  Algorithm Exgen(P, S, F)
2      if |P ∪ S| = k then
3          output P ∪ S
4          return True
5      hasIntLeaf ≔ False
6      p ≔ choose element of P
7      X ≔ N(p) \ (P ∪ S ∪ F)
8      for i from k − |P ∪ S| down to 0 do
9          hasIntLeaf_i ≔ False
10         foreach M ⊆ X such that |M| = i do
11             if Exgen(P ∪ M \ {p}, S ∪ {p}, F ∪ X)= True then
12                 hasIntLeaf_i ≔ True; hasIntLeaf ≔ True
13         if hasIntLeaf_i = False then
14             return hasIntLeaf
15                                  ▷ Stop recursion if no new solution was found
16     return hasIntLeaf
```

choice, since a solution does not need to contain a neighbor of p.

Next, we introduce a similar pruning rule as we did it for `Simple` and `Pivot`. To apply the rule, we enumerate all subsets M of the possible neighbors X in the following way: We start by creating children for subsets of size $i \coloneqq k - |P \cup S|$. If none of the children which correspond to these choices for M leads to an interesting leaf, then we prune the enumeration tree, that is, we return to the parent of the current search tree node. Otherwise, we decrease the size i of the subsets M of X that we want to generate by one and continue.

Proposition A.1. *The pruning rule performed in Lines 13–15 of Algorithm A.1 is correct.*

Proof. Consider a node T in the enumeration tree with vertex sets P_T, S_T, and F_T and current pivot vertex p. Furthermore, let X_T be the set of neighbors of vertex p which are not in $P_T \cup S_T \cup F_T$. Now assume that for some size m, for each size m set $M \subseteq X_T$, the recursive call for $P_T \cup M$ does not output any solution. Now consider a child R of T for which $|P_R| - |P_T| = m - 1$ holds, that is, P_R is obtained by adding a set M_R of size $m - 1$ to P_T. By the choice of m, there exists some child Q of T obtained by adding M_Q to P_T such that $M_Q \setminus M_R = \{v\}$ for some vertex v and Q does not lead to an interesting leaf. Now suppose that R leads to

an interesting leaf. Consider a vertex $u \notin M_R \cup P_T$ that is a leaf of some spanning tree of the corresponding subgraph. Removing u and adding the vertex v to this subgraph gives a connected subgraph that has to be enumerated in the enumeration subtree rooted at Q. This contradicts that Q does not lead to an interesting leaf. Hence, node R cannot lead to an interesting leaf. Consequently, no child obtained by adding a set M of size $m-1$ to P_T leads to an interesting leaf. By applying this argument inductively, we have that no child obtained by adding a set M of size smaller than m leads to an interesting leaf. □

This pruning rule is much weaker since it does not lead to a polynomial delay: Consider a node T in the enumeration tree. Furthermore, let X_T be the set of possible neighbors, where $|X_T| \leq \Delta$. The first time that we may return to the parent of T is if no branch for $M \subseteq X$ with $|M| = k - |P \cup S| - 1$ leads to an interesting leaf. There may be $\Theta(\Delta^{k-1})$ possibilities for choosing M which is not polynomial if k is not a constant. Nevertheless, the pruning rule proved very useful in the case of large k.

The following running time bound was observed by Komusiewicz and Sorge [150] and is stated only for the sake of comparison with the other running time bounds.

Theorem A.2 ([150])**.** ***Enumerate*** *with* ***Exgen*** *enumerates each connected subgraph of order at most k exactly once and has a worst-case running time of $\mathcal{O}((e(\Delta - 1))^{k-1} \cdot (\Delta + k) \cdot n/k$ time.*

A.2 Kavosh

The next algorithm in our comparison is `Kavosh` [127]. It was introduced for the computation of network motifs and is in some sense a mixture of `Simple` and `Exgen`; the pseudocode is shown in Algorithm A.2.

In each search tree node, we have the sets P, S, and F as defined in `Pivot` and `Exgen`, that is, P contains the vertices of the subgraph set whose neighbors may still be added, S contains the other vertices of the subgraph set, and F contains the vertices which may not be added to the subgraph anymore. The basic idea of `Kavosh` is that instead of choosing one pivot vertex, we extend the subgraph set by creating all possible subsets of the neighborhood of P. In other words, we now determine the set X of neighbors of P which are not in $P \cup S \cup F$. Then, for each non-empty $M \subseteq X$ of size at most $k - |P \cup S|$, we call `Kavosh` recursively with M being the vertex set whose neighbors are now considered, P being added to S, and with X being added to F, since in this child aim to enumerate those subgraphs extending $P \cup S$ that contain all vertices of M and no further vertices of X.

Algorithm A.2: The `Kavosh` algorithm; the initial call is `Kavosh`$(\{v\}, \emptyset, \emptyset)$.

```
1  Algorithm Kavosh(P, S, F)
2      if |P ∪ S| = k then
3          output P ∪ S
4          return True
5      hasIntLeaf = False
6      X := N(P) \ (P ∪ S ∪ F)
7                          ▷ Add at least one vertex to the subgraph
8      for i from k − |P ∪ S| down to 1 do
9          hasIntLeaf_i := False
10         foreach M ⊆ X such that |M| = i do
11             if Kavosh(M, S ∪ P, F ∪ X)= True then
12                 hasIntLeaf_i := True; hasIntLeaf := True
13         if hasIntLeaf_i = False then
14             return hasIntLeaf
15                                 ▷ Stop recursion if no new solution was found
16     return hasIntLeaf
```

We provide a similar pruning rule for `Kavosh` as we did it for `Exgen`. That is, we create the sets M in decreasing order of size; if for some size M, we do not obtain interesting leafs for any of the recursive calls, then we return immediately to the parent of the current search tree node. The proof of correctness of this pruning rule follows similar arguments as the proof for `Exgen`.

Proposition A.3. *The pruning rule performed in Lines 13–15 of Algorithm A.2 is correct.*

Proof. Consider a node T in the enumeration tree with vertex sets P_T, S_T, and F_T. Furthermore, let X_T be the set of neighbors of P_T which are not in $P_T \cup S_T \cup F_T$. Furthermore, consider some m such that for each subset M of X_T of size m the recursive call of `Kavosh`$(M, P_T \cup S_T, F_T \cup X_T)$ does not lead to an interesting leaf. We show that any recursive call for a set $M' \subseteq X$ of size less than m does not lead to interesting leaves. Let R be a child of T which was obtained by such a recursive call and assume R leads to an interesting leaf. Clearly, node T contains a child L created by the recursive call `Kavosh`$(M, P_T \cup S_T, F_T \cup X_T)$ where $M' \subsetneq M$ where $|M| = m$. Since L does not lead to an interesting leaf and $P_R \subset P_L$, $S_L = S_R$, and $F_L = F_R$ we have that R cannot lead to an interesting leaf. □

As in the case of `Exgen`, this pruning rule does not make `Kavosh` a polynomial delay algorithm for E-CISE. For example consider the star graph with one vertex v of degree $n-1$ and assume v is added in the root of the enumeration tree. After trying all subsets of size $k-1$ of $N(v)$, the algorithm tries to add each subset of size $k-2$, none of which gives a solution. The number of these subsets is $\binom{n-1}{k-2}$ which is not polynomial if k is not a constant. Hence, there is a superpolynomial delay between the output of the last solution and the termination of the algorithm. Nevertheless, in the experiments, this pruning rule proved to be critical in the case of large k.

We conclude by bounding the overall running time of `Enumerate` with `Kavosh`.

Lemma A.4. *`Enumerate` with `Kavosh` has a worst-case running time of* $\mathcal{O}((e(\Delta-1))^{(k-1)} \cdot \Delta \cdot n)$.

Proof. `Enumerate` with `Kavosh` enumerates each connected subgraph of order at most k exactly once [127]. This implies that for each pair of different nodes T and Q in the enumeration tree with the respective sets P_T, S_T, P_Q, and P_T we have that $P_T \cup S_T \neq P_Q \cup S_Q$. That is, each enumeration tree node corresponds to a different connected subgraph of order at most k. According to Lemma 3.1, the overall number of nodes in the enumeration trees over all calls to `Kavosh` is $\mathcal{O}((e(\Delta-1))^{(k-1)} \cdot (n/k))$.

It remains to bound the time per node T in the enumeration tree. Determining the neighbors X_T of the subgraph set P_T needs $\mathcal{O}(k\Delta)$ time since $|P_T| \leq k$ and each vertex has up to Δ neighbors. For each subset M of X_T of size at most $k-|P_T \cup S_T|$ we make a recursive call with parameters $M, P_T \cup S_T, F_T \cup X_T$. The recursive call includes computing the three sets which can be done in $\mathcal{O}(k\Delta)$ time per call. By charging this running time to the corresponding child in the enumeration tree, we obtain a running time of $\mathcal{O}(k\Delta)$ per enumeration tree node. Outputting a solution needs $\mathcal{O}(k)$ time. Hence, we obtain a delay of $\mathcal{O}(k\Delta)$ per node in the enumeration tree. The overall running time follows. □

A.3 BDDE: Breadth-First Discovery, Depth-First Extension

A furthermore, seemingly more involved, enumeration algorithm in our comparison is `BDDE` [167]. The pseudocode of `BDDE` is shown in Algorithm A.3. The idea is to start with a vertex v and to enumerate all subgraphs of order at most k such that each connected subgraph S that contains v is enumerated before S' if $S \subseteq S'$.

This algorithm was used to enumerate all connected subsets S such $f(S) \geq t$ for a given function f and a fixed threshold t. In our case of enumerating all connected subgraphs of order k, $f(S) = |S|$ and $t = k$.

To enumerate for a given vertex v all connected induced subgraphs of order at most k containing v in the order of their inclusion relation, the algorithm will copy parts of the search tree. The algorithm consists of two functions: `Depth` to discover new vertices, and `Breadth` to copy parts of the search tree.

In the enumeration tree T each node has a label id which represents exactly one vertex in the graph G and each path from the root to a another node in the tree represents a connected subgraph in G. Hence, the set of leaves in depth k corresponds to the set of connected subgraphs of order k. Clearly, the root has id v. Consider node x with id u in the enumeration tree. The set P is referred to as the set of ids of the nodes in the enumeration tree which are predecessors of x. The *exclusive neighborhood* $X(u) := N(u) \setminus N[P]$ is the set of neighbors of u in the graph G which are neither in P nor neighbors of the vertices in P and not in the set P.

Clearly, a sibling s of a node t in the search tree has many children similar to children of t. The algorithm uses this fact as follows: Let r be a child of s. If r is not an exclusive neighbor of t, copy the subtree which is rooted at r and call this copy r'. Next, make r' to a child of t. This will be achieved by the procedure `Breadth`. Furthermore, procedure `Depth` will be used to add new edges and to discover new vertices.

Now, we describe the procedures `Depth` and `Breadth` in further detail.

`Depth` has three parameters: The set P, which is the set of all names of predecessors of p, the vertex u, which is the actual vertex we consider, and a list C which stores all successors of nodes representing the last vertex in P. First, we make a new node x in the enumeration tree with the name u and we create a new list C' for the branches of x. Second, with `Breadth` we copy the branches stored in C as new branches of x. If we copy a branch x' successfully we add an edge from x to x' and append x' to C'. Third, for each vertex $z \in X(u)$ we call `Depth`. If we created a branch x' successfully we create an edge from x to x' and append x' to C'. In the end we return x.

`Breadth` has three parameters: The set P is the set of all names of predecessors of node x, the node x which is a sibling of the node this function will create, and the set X is the exclusive neighborhood of the last node in the set P created from the function `Depth`. The set X is needed, to avoid enumerating some subgraphs twice. If the name of x is in the set X, we return. If not returned, we create a new node x' which has the same name as node x. Then for each successor y of x in T, we call `Breadth` recursively. If this call returns a node x^*, we make an edge from x' to x^*.

Algorithm A.3: The `BDDE` algorithm. Here, $Tree$ is the enumeration tree which is initially empty. The initial call is `Depth`$([\,], v, [\,])$.

```
1  Procedure Depth(P, u, C)
2  |  X := N(u) \ N[P]; C' := [ ]
3  |  P := P ∪ {u}
4  |  x := Tree.number_of_vertices
5  |  Tree.add_vertex(x, id = u)
6  |  if |P| = k then
7  |  |  output P
8  |  |  return −1
9  |  for y ∈ C do
10 |  |  x' := Breadth(P, y, X)
11 |  |  if x' ≠ −1 then
12 |  |  |  Tree.add_edge(x, x')
13 |  |  |  C'.append(x')
14 |  for z ∈ X do
15 |  |  x' := Depth(P, z, C')
16 |  |  if x' ≠ −1 then
17 |  |  |  Tree.add_edge(x, x')
18 |  |  |  C'.append(x')
19 |  return x
20 Procedure Breadth(P, x, X)
21 |  if Tree.vertex(x)[id] ∈ X then
22 |  |  return −1
23 |  P := P ∪ {Tree.vertex(x)[id]}
24 |  if |P| := k then
25 |  |  output P
26 |  |  return −1
27 |  x' := Tree.number_of_vertices
28 |  Tree.add_vertex(x', id = id(x))
29 |  for y ∈ Tree.successor(x) do
30 |  |  x* := Breadth(P, y, X)
31 |  |  if x* ≠ −1 then
32 |  |  |  Tree.add_edge(x', x*)
33 |  return x'
```

In the end of this function, we return x'.

Now, we bound the running time of `BDDE`.

Lemma A.5. *`Enumerate` with `BDDE` has a worst-case running time of* $\mathcal{O}((e(\Delta - 1))^k \cdot k \cdot \Delta \cdot n)$ *time.*

Proof. It was shown in [167, Lemma 5] that for two nodes T and S in the enumeration trees of `Enumerate` with `BDDE` the vertex sets P_T and P_S are different. In other words, each subgraph of size at most k is enumerated exactly once.

We now bound the running time for each call of the procedures `Depth` and `Breadth`. We start with `Depth` with parameters P, u, and C. Since each vertex has degree at most Δ, the size of the exclusive neighborhood X of u is bounded by $\mathcal{O}(\Delta)$. By marking each vertex of $N[P]$ with a color, we can determine the exclusive neighborhood X in $\mathcal{O}(\Delta)$ time. In each recursive call of `Depth` we enlarge C by at least Δ, the size of X. Hence, there are at most $k\Delta$ recursive calls of `Breadth`. Each recursive call of `Breadth` includes computing the new vertex which can be done in $\mathcal{O}(\Delta)$ time. We charge this running time to the corresponding child in the enumeration tree. Furthermore, there are at most Δ calls of `Depth`. Similar, each recursive call of `Depth` includes computing the new vertex which can be done in $\mathcal{O}(\Delta)$ time. Again, we charge this running time to the corresponding child. Overall, `Depth` needs $\mathcal{O}(\Delta)$ time to construct the next child.

Now, we consider `Breadth` with parameters P, x, and X. Checking existence of the name of node x in the set X can be done in constant time. Since each vertex in the graph has degree at most Δ, there are at most Δ recursive calls for `Breadth`. Again, we charge the running time of each recursive call to the corresponding children. Overall, `Breadth` needs $\mathcal{O}(\Delta)$ time to construct the next child.

Since each solution can be output in $\mathcal{O}(k)$ time, we obtain a running time of $\mathcal{O}(k+\Delta)$ per enumeration tree node. The overall running time follows. □

Since the basic idea of algorithm `BDDE` is to start with a vertex v and to enumerate all connected subgraphs of order at most k such that each connected subgraph S which contains v is enumerated before S' if $S \subseteq S'$, algorithm `BDDE` does *not* yield a polynomial delay for E-CISE. For example, consider the case $k = n$ and G is complete. Then `BDDE` enumerates each connected induced subgraph of order less than k, before enumerating the graph G. Clearly, these are exponentially many.

A.4 RSSP

The last algorithm in our comparison is `RSSP` [4]. It was introduced to enumerate all connected induced subgraphs and to mine all maximal cohesive subgraphs. The

Algorithm A.4: The `RSSP` algorithm; the initial call is `RSSP`$(\{v\}, N(v))$.

```
1  Algorithm RSSP(P, X)
2      s ≔ first vertex of P
3      z ≔ last vertex of P
4      if |P| = k then
5          output P
6          return
7      while X ≠ ∅ do
8          u ≔ choose first vertex from X
9          delete u from X
10         remove invalid candidates from X
11         for w ∈ N(u) do
12             if dist(s, w) > dist(s, z) or (dist(s, w) = dist(s, z) and w > z)
                then
13                 X ≔ X ∪ {w}
14                 RSSP(P ∪ {u}, X)
15     return
```

pseudocode is shown in Algorithm A.4. Each search tree node T_i consists of a subgraph set P and a set X of vertices which can be added to P. Fix an ordering on the vertices of V. For $x, y \in P$ by $\text{dist}(x, y)$ we denote the length of a shortest path from x to y in $G[P]$. Let $z \in P$ be the vertex with smallest label with respect to that ordering, and let $W \subseteq P$ be the set of vertices of P which have the longest shortest path in $G[P]$ to vertex v. Let $u \in W$ be the vertex with maximal index. The subgraph set of the parent T_{i-1} of node T_i is $P \setminus \{u\}$. Let $w \in N(P)$. If $\text{dist}(z, w) > \text{dist}(u, w)$ or $\text{dist}(z, w) = \text{dist}(u, w)$ and the index of w is larger than the index of u, then node T_i has a child with subgraph set $P \cup \{w\}$. To obtain the search tree nodes T_{i+1} and T_{i-1} efficiently, all distances to vertex v and all parents are saved. Due to the strict child definition, a vertex $x \in X_i$ cannot be a valid candidate to expand P_{i+1}. To this end, after adding vertex p_i to P_i to obtain P_{i+1}, all vertices in X_i are checked if they are still a valid candidate to expand P_i. Because of this step, a similar pruning rule as introduced for `Simple` and `Pivot` is not possible since the ordering of X changes during the exploration of the search tree.

Appendix B

Details for FixCon

B.1 Implementation of Objective Functions

To demonstrate the ease-of-use of *FixCon*, we present the source code of the methods for setting the function parameters and for computing the objective functions.

```
def set_problem(prob, paras, k):
     global problem
     global parameters
     global edge_monotone
     global vertex_upper_bound
     global user_ub
     problem = Fco(prob)
     parameters = paras
     # densest k-subgraph
     if problem == Fco.DENSEST:
          edge_monotone = True
          vertex_upper_bound = k
          user_ub = k*(k-1)/2
     # maximize min-degree
     elif problem == Fco.MINDEG:
          edge_monotone = True
          vertex_upper_bound = 1
          user_ub = k-1
     # minimize max-degree
     elif problem == Fco.MINMAXDEG:
          edge_monotone = False
```

```
          vertex_upper_bound = 0
          user_ub = -2
     # find an acyclic graph
     elif problem == Fco.ACYCLIC:
          edge_monotone = False
          vertex_upper_bound = 0
          user_ub = 1
     # find a triangle free graph
     elif problem == Fco.TRIAFREE:
          edge_monotone = False
          vertex_upper_bound = 0
          user_ub = 1
     # maximize the diameter
     elif problem == Fco.MAXDIAM:
          edge_monotone = False
          vertex_upper_bound = 1
          user_ub = k-1
     # find r-regular graph
     elif problem == Fco.RREG:
          edge_monotone = False
          vertex_upper_bound = 1
          user_ub = 1
     # find graph with min-deg >= alpha and max-deg <= beta
     elif problem == Fco.BOUNDDEG:
          edge_monotone = False
          vertex_upper_bound = 1
          user_ub = 1



def evaluate(graph):
     global problem
     global parameters
     if problem == Fco.DENSEST: # densest k-subgraph
          return graph.ecount()
     elif problem == Fco.MINDEG: # maximize min-degree
          min_deg = graph.vcount()
          for i in range(graph.vcount()):
```

```
            min_deg = min(graph.degree(i),min_deg)
        return min_deg
    elif problem == Fco.MINMAXDEG: # minimize max-degree
        return -graph.maxdegree()
    elif problem == Fco.ACYCLIC: # find an acyclic graph
        if graph.ecount() == graph.vcount() - 1:
            return 1
        else:
            return 0
    elif problem == Fco.TRIAFREE: # find a triangle free graph
        for edge in graph.es:
            # check if neighborhoods of endpoints are not disjoint
            if not set(graph.neighbors(edge.source)).
                      isdisjoint(set(graph.neighbors(edge.target))):
                return 0
        return 1
    elif problem == Fco.MAXDIAM: # maximize the diameter
        return graph.diameter(directed=False)
    elif problem == Fco.RREG: # find r-regular graph
        # r=parameters[0]
        if graph.maxdegree() > parameters[0]:
            return -sys.maxint
        for i in range(graph.vcount()):
            if parameters[0] > graph.degree(i):
                return 0
        return 1
    elif problem == Fco.BOUNDDEG: # find graph with
        # min-deg >= alpha and max-deg <= beta
        alpha=parameters[0], beta=parameters[1]
        if graph.maxdegree() > parameters[1]:
            return -sys.maxint
        for i in range(graph.vcount()):
            if parameters[0] > graph.degree(i):
                return 0
        return 1
```

B.2 Details for the ILP formulations

Densest Subgraph. We introduce a binary variable $y_{\{u,v\}}$ for each edge $\{u, v\} \in E$. The objective is

$$\text{maximize} \sum_{\{u,v\} \in E} y_{\{u,v\}}$$

subject to the constraints

$$\begin{aligned} y_{\{u,v\}} &\leq x_v && \forall \{u, v\} \in E, \\ y_{\{u,v\}} &\leq x_u && \forall \{u, v\} \in E. \end{aligned}$$

We set the BestObjStop parameter of Gurobi to $\binom{k}{2}$ so that it may stop after finding a clique of size k.

Max Min Degree. We introduce an additional variable y_δ. The objective is

$$\text{maximize } y_\delta$$

subject to the constraints

$$k \cdot (1 - x_v) + \sum_{u \in N(v)} x_u \geq y_\delta \qquad \forall v \in V$$

We set the BestObjStop parameter of Gurobi to $k - 1$ so that it may stop after finding a clique of size k.

Min Max Degree. We introduce an additional variable y_Δ. The objective is

$$\text{minimize } y_\Delta$$

subject to the constraints

$$k \cdot (1 - x_v) + \sum_{u \in N(v)} x_u \leq y_\Delta \qquad \forall v \in V$$

We set BestObjStop=2 so that Gurobi may stop after finding a graph with maximum degree 2.

Acyclic Subgraph. For each vertex of v that has at least two neighbors, we pick two arbitrary neighbors u and w of v and check if u and w are adjacent. If this is the case, we add the triangle constraint

$$x_u + x_v + x_w \leq 2.$$

Then, we add further cycle constraints in a lazy manner in callbacks. That is, at a search tree node where some solution S has been computed, we compute the cycle basis of the graph $G[S]$ and for each cycle C in the cycle basis, we add the constraint

$$\sum_{v \in C} x_v \leq |C| - 1.$$

The graph has a solution if and only if the ILP is feasible.

Triangle-Free Subgraph. For each vertex of v that has at least two neighbors, we pick two arbitrary neighbors u and w of v and check if u, v, and w form a triangle. If this is the case, we add the triangle constraint

$$x_u + x_v + x_w \leq 2.$$

Further triangle constraints are added in a lazy manner, whenever an intermediate solution contains some triangles. The graph has a solution if and only if the ILP is feasible.

Maximum-Diameter Subgraph. This problem was the most difficult to formulate. As finding a subgraph of diameter at least 2 is trivial for these values of k, we assume that we are searching for subgraphs with diameter at least 3. This allows adding the constraint

$$k \cdot x_v + \sum_{u \in N(v)} x_u \leq 2k - 2 \qquad \forall v \in V$$

which enforces that the solution does not contain a vertex v plus $k-1$ neighbors of v. To model the diameter-maximization we add a variable y_{diam} and set the objective to

$$\text{maximize } y_{\text{diam}}.$$

The constraints ensuring that y_{diam} is the diameter of the solution are added in a lazy manner. After computing an intermediate solution S, we compute the diameter d_S of $G[S]$, and if $G[S]$ is connected, that is, $d_S \leq \infty$ we add the constraint

$$y_{\text{diam}} + \sum_{v \in S} k \cdot x_v \leq k^2 + d_S$$

ensuring that $y_{\text{diam}} \leq d_S$ if the solution equals S.

r-Regular Subgraph. We add the constraints

$$-rx_v + \sum_{u \in N(v)} x_u \geq 0 \qquad \forall v \in V,$$
$$k \cdot x_v + \sum_{u \in N(v)} \leq k + r \qquad \forall v \in V$$

fixing the degree of every selected vertex to r. The graph has a solution if and only if the ILP is feasible.

(α, β)-Degree-Constrained Subgraph. We add the constraints

$$-\alpha x_v + \sum_{u \in N(v)} x_u \geq 0 \qquad \forall v \in V,$$
$$k \cdot x_v + \sum_{u \in N(v)} \leq k + \beta \qquad \forall v \in V$$

fixing the degrees of every selected vertex to be in $\{a, \ldots, b\}$. The graph has a solution if and only if the ILP is feasible.

B.3 Further Experimental Results for FixCon

Table B.1 shows for each instance the largest value k^* such that all eight problems could be solved for all $k \leq k^*$. `Pivot` and `Simple` are the respective algorithms with all improvements up to the heuristic lower bound. `Pivot+` has additionally the problem-specific lower bounds described in Section 4.7. A value of N/A for the ILP means that for all k, there is at least one problem which the ILP did not solve for this graph.

Table B.1: Detailed results of our algorithm compared with ILPs.

Size	Name	$\|V\|$	$\|E\|$	`Pivot`	`Simple`	`Pivot+`	`ILP`
Small	moreno-zebra	27	111	**20**	**20**	**20**	**20**
	ucidata-zachary	34	78	15	14	**15**	9
	contiguous-usa	49	107	12	12	18	**20**
	dolphins	62	159	9	9	**18**	15
	ca-sandi-auths	86	124	11	12	**18**	15
	adjnoun_adjacency	112	425	6	6	12	**17**
	arenas-jazz	198	2 742	6	6	**20**	16
	inf-USAir97	332	2 126	6	5	**17**	10
	ca-netscience	379	914	8	8	**19**	13
	bio-celegans	453	2 025	6	5	**13**	11
Medium	bio-diseasome	516	1 188	7	6	**20**	12
	soc-wiki-Vote	889	2 914	6	5	12	**15**
	arenas-email	1 133	5 451	6	5	13	**16**
	inf-euroroad	1 174	1 417	10	10	13	**18**
	bio-yeast	1 458	1 948	7	7	12	**14**
	ca-CSphd	1 882	1 740	9	10	**13**	10
	soc-hamsterster	2 426	16 630	6	5	**20**	12
	inf-openflights	2 939	15 677	6	5	**16**	8
	ca-GrQc	4 158	13 422	6	5	**20**	12
	inf-power	4 941	6 594	8	8	17	**18**
Large	soc-advogato	6 541	39 432	6	4	**11**	6
	bio-dmela	7 393	25 569	6	4	**8**	5
	ca-HepPh	11 204	117 619	6	5	**20**	13
	ca-AstroPh	17 903	196 972	6	5	**17**	N/A
	soc-brightkite	56 739	212 945	5	4	**15**	N/A
	coAuthorsCiteseer	227 320	814 134	5	5	**20**	N/A
	coAuthorsDBLP	299 067	977 676	5	5	**16**	N/A
	soc-twitter-follows	404 719	713 319	4	4	**5**	N/A
	coPapersCiteseer	434 102	16 036 720	6	5	**20**	N/A
	coPapersDBLP	540 486	15 245 729	5	5	**13**	N/A
Dense	bn-cat-mixed-species_brain_1	65	730	7	6	**14**	11
	robot24c1_mat5	404	14 261	12	6	11	**13**
	econ-beause	507	39 428	6	5	20	6
	bn-mouse_retina_1	1 076	90 811	6	5	**16**	12
	comsol	1 500	48 119	6	5	**20**	N/A
	bn-fly-drosophila_medulla_1	1 781	8 911	6	4	**10**	5
	heart2	2 339	340 229	8	6	**20**	9
	econ-orani678	2 529	86 768	6	6	**11**	N/A
	psmigr_1	3 140	410 781	6	5	**17**	16
	bn-human-BNU_1_0025890	177 584	15 669 037	5	5	**9**	N/A