FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

# On the Complexity of Local Search Problems with Scalable Neighborhoods

**Dissertation**
**zur Erlangung des akademischen Grades**
**doctor rerum naturalium (Dr. rer. nat.)**

**vorgelegt dem Rat der Fakultät für Mathematik und Informatik**
**der Friedrich-Schiller-Universität Jena**

**von Nils Morawietz, M.Sc.**
**geboren am 19. Juli 1995 in Marburg, Deutschland.**

Gutachter:
Prof. Dr. Christian Komusiewicz, Friedrich-Schiller-Universität Jena, Deutschland
Prof. Dr. Peter Rossmanith, RWTH Aachen, Deutschland
Prof. Dr. Stefan Szeider, Technische Universität Wien, Österreich

Tag der öffentlichen Verteidigung: 21. Oktober 2024

## Ehrenwörtliche Erklärung

Hiermit erkläre ich,

- dass mir die Promotionsordnung der Fakultät bekannt ist,
- dass ich die Dissertation selbst angefertigt habe, keine Textabschnitte oder Ergebnisse eines Dritten oder eigenen Prüfungsarbeiten ohne Kennzeichnung übernommen und alle von mir benutzten Hilfsmittel, persönlichen Mitteilungen und Quellen in meiner Arbeit angegeben habe,
- dass ich die Hilfe eines Promotionsberaters nicht in Anspruch genommen habe und dass Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen,
- dass ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts haben mich folgende Personen unterstützt:

Jaroslav Garvardt, Niels Grüttemeier, Christian Komusiewicz, Simone Linz, André Nichterlein, Jannik Schestag, Mathias Weller.

Ich habe die gleiche, eine in wesentlichen Teilen ähnliche bzw. eine andere Abhandlung noch bei keiner anderen Hochschule als Dissertation eingereicht.

Ort, Datum Nils Morawietz, Unterschrift

Morawietz, Nils
*On the Complexity of Local Search Problems with Scalable Neighborhoods*
Dissertation, Friedrich-Schiller-Universität Jena, 2024.

## Curriculum vitae

- Since May 2023: Member of the *Algorithm Engeneering* group
  Friedrich-Schiller-Universität Jena, Germany.
- October 2019 – April 2023: Member of the *Algorithmics* group
  Philipps-Universität Marburg, Germany.
- October 2017 – September 2019: M.Sc. in *Computer Science*
  Philipps-Universität Marburg, Germany.
- October 2013 – September 2017: B.Sc. in *Computer Science*
  Philipps-Universität Marburg, Germany.

# Preface

This thesis summarizes my research findings on the complexity of local search algorithms with scalable neighborhoods. The results contained in this thesis were obtained from April 2020 to April 2023 at the Philipps-Universität Marburg at the Fachbereich Mathematik und Informatik in the Algorithmics research group lead by Christian Komusiewicz and from May 2023 to January 2024 at the Friedrich-Schiller-Universität Jena at the Institut für Informatik in the Algorithm Engeneering research group lead by Christian Komusiewicz.

In the following, I describe which chapters are based on which previous publications and highlight my contribution to these publications.

Chapter 3 is based on the publication "Parameterized Local Search for Vertex Cover: When Only the Search Radius Is Crucial" written with Christian Komusiewicz. A preliminary version of this publication appeared in the *Proceedings of the 17th International Symposium on Parameterized and Exact Computation (IPEC '22)* [107]. Christian proposed to analyze the parameterized complexity of local search for Weighted Vertex Cover. In particular, the goal was to develop algorithms for this local search problem that run in $\ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time, where $\ell$ is some secondary parameter that ideally fulfills $\ell \ll n$. The first parameter we considered was the maximum degree $\Delta$ of the input graph. We showed that such an algorithm for $\ell = \Delta$ is possible, which directly generalized a previously known algorithm for the unweighted version of Vertex Cover [98]. Afterwards, we also developed algorithms with similar running times for other structural parameters $\ell$. All results of this chapter were developed by both coauthors together. I worked out all technical details and prepared the draft of the manuscript.

Chapter 4 is based on the publication "Parameterized Local Search for Max $c$-Cut" written with Jaroslav Garvardt, Niels Grüttemeier, and Christian Komusiewicz. A preliminary version of this publication appeared in the *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI '23)* [66]. To experimentally evaluate the performance of parameterized local search for classical graph problems, Christian and I decided to analyze local search for Max Cut and the more general Max $c$-Cut. All authors jointly developed an algorithm solving the problem in $(3 \cdot e \cdot \Delta)^k \cdot n^{\mathcal{O}(1)}$ time. I worked out all the technical details and implemented this algorithm. Niels and I developed several pruning rules to reduce the practical running time of the algorithm and Niels prepared the draft of these pruning rules. I performed the experimental evaluation of our algorithm with some assistance by Jaroslav. From a negative point of view, we discovered that the considered parameterized local search problem is W[1]-hard when parameterized by the

search radius $k$. In addition, after the publication of the conference version, I developed several further hardness results for local search versions of related partition problems.

Chapter 5 is based on the publication "Graph Clustering Problems Under the Lens of Parameterized Local Search" written with Jaroslav Garvardt, André Nichterlein, and Mathias Weller. A preliminary version of this publication appeared in the *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC '23)* [67]. Based on the huge success of local search solvers during the PACE-challenge for Cluster Editing, I proposed to analyze the parameterized complexity of local search for Cluster Deletion and Cluster Editing with respect to the move neighborhood. We initially observed that the problems are FPT with respect to the maximum degree $\Delta$ plus the search radius $k$. André and I worked out the technical details of this algorithm and André prepared an initial draft of this algorithm and proved that it runs in $\Delta^{8k} \cdot k^k \cdot n^{\mathcal{O}(1)}$ time. After the publication of the conference version, I was able to present a different algorithm that solves the problems in $\Delta^{2k} \cdot (3 \cdot e)^k \cdot n^{\mathcal{O}(1)}$ time. Additionally, all authors jointly discovered that the local search version of Cluster Deletion is W[1]-hard when parameterized by the search radius $k$. Jaroslav and I worked out the technical details and prepared the draft of this hardness result together. Furthermore, I discovered several hardness results for LS Cluster Editing and developed FPT-algorithms for the permissive versions of both local search problems. For all these results, I worked out the technical details and prepared the draft of the manuscript.

Chapter 6 is based on the publication "On the Complexity of Parameterized Local Search for the Maximum Parsimony Problem" written with Christian Komusiewicz, Simone Linz, and Jannik Schestag. A preliminary version of this publication appeared in the *Proceedings of the 34th Annual Symposium on Combinatorial Pattern Matching (CPM '23)* [104]. Initially, we aimed to analyze the (parameterized) complexity of local search for Maximum Parsimony with respect to the $d_{\text{ECR}}$-neighborhoods. We observed that for the $d_{\text{sECR}}$-neighborhood, an FPT-algorithm with respect to the search radius $k$ is possible. Jannik and I worked out the technical details together. Later, Christian, Simone, and I observed that the parameterized complexity changes, when considering the $d_{\text{ECR}}$-neighborhood. Namely, we observed that the problem then becomes W[1]-hard when parameterized by the search radius $k$. I was later able to extend this result to other famous neighborhoods for Maximum Parsimony and even if each character is binary. Furthermore, after the publication of the conference version I was able to lift the intractability results to the permissive version of the respective local search problems. I worked out all the details for these intractability results and prepared the draft of the manuscript.

Chapter 7 is based on the publication "Finding 3-Swap-Optimal Independent Sets and Dominating Sets Is Hard" written with Christian Komusiewicz. A preliminary version of this publication appeared in the *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS '22)* [106] and a full version is accepted in principle at *ACM Transactions on Computation Theory*. Christian proposed to consider the complexity for finding locally optimal solutions for Weighted Independent Set with respect to the $k$-swap neighborhood for small values of $k$. Initially, I found a reduction proving PLS-hardness for each $k \geq 7$. Later, I was able to lift this hardness even to all values of $k \geq 3$. Furthermore, after the publication of the conference version we observed that this hardness result could also be shown for a large class of weighted optimization problems. Based on these results, Christian raised the question about the complexity of finding locally optimal solutions for $k \leq 2$. Christian and I jointly developed polynomial-time algorithms for this case. I worked out all the technical details of all results and prepared the draft of the manuscript.

Next, I list all my other conference and journal publications I worked on during my PhD-studies. These publications are ordered alphabetically.

- "A cop and robber game on edge-periodic temporal graphs", with Thomas Erlebach, Jakob T. Spooner, and Petra Wolf. Journal: *Journal of Computer and System Sciences* [49].

- "A Graph-Theoretic Formulation of Exploratory Blockmodeling", with Alexander Bille, Niels Grüttemeier, and Christian Komusiewicz. Conference: *21st International Symposium on Experimental Algorithms (SEA '23)* [18].

- "A Timecop's Chase Around the Table", with Petra Wolf. Journal: Part of a joint journal version [49]. Conference: *46th International Symposium on Mathematical Foundations of Computer Science (MFCS '21)* [128].

- "A Timecop's Work Is Harder Than You Think", with Carolin Rehs and Mathias Weller. Conference: *45th International Symposium on Mathematical Foundations of Computer Science (MFCS '20)* [127].

- "Can Local Optimality Be Used for Efficient Data Reduction?", with Christian Komusiewicz. Journal: Submitted. Conference: *12th International Conference on Algorithms and Complexity (CIAC '21)* [105].

- "Complexity of Local Search for Euclidean Clustering Problems", with Bodo Manthey, Jesse van Rhijn, and Frank Sommer. ArXiv: [121].

- "Distance to Transitivity: New Parameters for Taming Reachability in Temporal Graphs", with Arnaud Casteigts and Petra Wolf. Conference: *49th International Symposium on Mathematical Foundations of Computer Science (MFCS '24)* [29].

- "Efficient Bayesian Network Structure Learning via Parameterized Local Search on Topological Orderings", with Niels Grüttemeier and Christian Komusiewicz. Conference: *35th Annual Conference on Artificial Intelligence (AAAI '21)* [77].

- "Exact Algorithms for Group Closeness Centrality", with Luca Pascal Staus, Christian Komusiewicz, and Frank Sommer. Conference: *2nd SIAM Conference on Applied and Computational Discrete Algorithms (ACDA '23)* [157].

- "Multi-Parameter Analysis of Finding Minors and Subgraphs in Edge-Periodic Temporal Graphs", with Emmanuel Arrighi, Niels Grüttemeier, Frank Sommer, and Petra Wolf. Conference: *48th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '23)* [11].

- "On the Complexity of Community-aware Network Sparsification", with Emanuel Herrendorf, Christian Komusiewicz, and Frank Sommer. Conference: *49th International Symposium on Mathematical Foundations of Computer Science (MFCS '24)* [88].

- "On the Complexity of Computing Time Series Medians Under the Move-Split-Merge Metric", with Jana Holznigenkemper, Christian Komusiewicz, and Bernhard Seeger. Conference: *48th International Symposium on Mathematical Foundations of Computer Science (MFCS '23)* [89].

- "On the Parameterized Complexity of Polytree Learning", with Niels Grüttemeier and Christian Komusiewicz. Conference: *30th International Joint Conference on Artificial Intelligence (IJCAI '21)* [78].

- "Parameterized Algorithms for Multi-Label Periodic Temporal Graph Realization", with Thomas Erlebach and Petra Wolf. Conference: *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND '24)* [50].

- "Preventing Small $(s, t)$-Cuts by Protecting Edges", with Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer. Conference: *47th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '21)* [80].

- "String Factorizations Under Various Collision Constraints", with Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer. Conference: *31th Annual Symposium on Combinatorial Pattern Matching (CPM '20)* [79].

**Acknowledgements.** I want to express my gratitude to Christian Komusiewicz who accepted me as his PhD student and introduced me to the world of parameterized complexity. No matter the time or date, Christian was always available for discussion and new ideas. My thanks also goes to my colleagues who always brought up new fascinating research topics and provided a pleasant working environment: Niels Grüttemeier, Frank Sommer, Jaroslav Garvardt, and Luca Pascal Staus. I want to thank all of my other co-authors: Petra Wolf, Tomas Erlebach, Mathias Weller, Emmanuel Arrighi, Kaja Balzereit, Alexander Bille, Emanuel Herrendorf, Jana Holznigenkemper, Simone Linz, Bodo Manthey, André Nichterlein, Carolin Rehs, Dennis Reinhardt, Jesse van Rhijn, Jannik Schestag, Bernhard Seeger, and Stefan Windmann. I am also grateful for the financial support by the Deutsche Forschungsgemeinschaft project Multivariate Operationale Parametrisierung für Heuristiken (OPERAH) during my PhD. Finally, I want to express my gratitude to my parents for their endless love and support over all the years!

# Abstract

Local search algorithms are heuristics that find a good solution to an instance of an optimization problem at hand by computing some starting solution and afterwards iteratively replacing the current solution by a better one in the local neighborhood. Here, the local neighborhood of the current solution is a collection of other solutions that are structurally similar to the current one. This process terminates, when a locally optimal solution is found, that is, when the current solution cannot be improved by any solution in its local neighborhood. The choice for the local neighborhood is a fundamental design aspect of such algorithms and highly influences the quality of the found solutions as well as the running time of the whole algorithm.

We study local search for several NP-hard optimization problems such as Vertex Cover and Max Cut and their arguably most natural neighborhoods. For these problems we consider the two most important questions about the worst-case running time of such local search algorithms: First, how fast can we find a better solution in the local neighborhood, if one exists? Second, how fast can we find a locally optimal solution? The local neighborhoods we consider are scalable, that is, these local neighborhoods grow with respect to some user-dependent parameter $k$ (called the search radius). The parameter $k$ determines a trade-off between solution quality and running time of the local search algorithm: the larger $k$, the better the final solution might be but the more time the algorithm takes. For unbounded values of $k$, all considered local search problems become NP-hard. Based on this NP-hardness, for the first question, we analyze the considered local search problems from the perspective of parameterized complexity with respect to the search radius $k$. As we show, for all these problems, determining whether there is a better solution in the local neighborhood needs $n^{\Omega(k)}$ time, assuming the Exponential Time Hypothesis. We complement these running time lower bounds by presenting algorithms with running times of the form $\ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ for some secondary parameter $\ell \leq n$. Regarding the second question involving the total running time of the local search algorithm, we consider a large class of weighted optimization problems where each feasible solution is a subset of some polynomial-size universe. As the local neighborhood of a solution $S$, we consider all solutions whose symmetric difference with $S$ has size at most $k$. We present a dichotomy with respect to $k$ on when locally optimal solutions can be found in polynomial time. Namely, we show that for $k \leq 2$, locally optimal solutions can be found in polynomial time, whereas finding a locally optimal solution in polynomial time is unlikely for each $k \geq 3$.

# Zusammenfassung

Lokale-Suche-Algorithmen sind Heuristiken, die gute Lösungen für ein Optimierungsproblem finden, indem sie zuerst eine Startlösung berechnen und anschließend wiederholt die jeweils aktuelle Lösung durch eine bessere Lösung in der lokalen Nachbarschaft ersetzen. Die lokale Nachbarschaft der aktuellen Lösung ist hierbei eine Menge anderer Lösungen, die der aktuellen Lösung strukturell ähnlich sind. Der Lokale-Suche-Algorithmus terminiert, wenn eine lokal optimale Lösung gefunden wurde, also eine Lösung, deren lokale Nachbarschaft keine bessere Lösung enthält. Die Wahl der lokalen Nachbarschaft ist ein fundamentaler Design-Aspekt von solchen Algorithmen und hat einen hohen Einfluss auf die Qualität der final gefundenen Lösung sowie auf die Gesamtlaufzeit des Algorithmus.

In dieser Arbeit untersuchen wir Lokale-Suche für diverse NP-schwere Optimierungsprobleme (wie zum Beispiel Vertex Cover und Max Cut) bezüglich der natürlichsten Nachbarschaften für diese Probleme. Für diese Probleme betrachten wir die zwei wichtigsten Fragen bezüglich der Komplexität von solchen Lokale-Suche-Algorithmen: Erstens, wie schnell können wir eine bessere Lösung in der lokalen Nachbarschaft finden, sofern eine existiert? Zweitens, wie schnell können wir eine lokal optimale Lösung finden? Die lokalen Nachbarschaften, die wir betrachten sind skalierbar, das heißt, diese lokalen Nachbarschaften wachsen bezüglich eines vom Anwender definierten Parameters $k$ (dem sogenannten Suchradius). Der Parameter $k$ stellt eine Stellschraube zwischen Lösungsgüte und Laufzeit des Lokale-Suche-Algorithmus dar: Je größer $k$ wird, desto besser kann die finale Lösung sein, aber desto länger wird der Algorithmus brauchen. Für ausreichend große Werte von $k$ sind alle in dieser Arbeit untersuchten Lokale-Suche-Probleme NP-schwer. Basierend auf dieser NP-Schwere, untersuchen wir die erste Frage aus der Perspektive der parametrisierten Komplexität bezüglich des Suchradius $k$. Wir zeigen, dass für alle betrachteten Probleme die Bestimmung, ob eine bessere Lösung in der lokalen Nachbarschaft existiert, mindestens $n^{\Omega(k)}$ Zeit benötigt, sofern die Exponentialzeit-Hypothese stimmt. Wir komplementieren diese untere Schranke für die Laufzeit, indem wir Algorithmen präsentieren, deren Laufzeit die Form $\ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ haben, für sekundäre Parameter $\ell \leq n$. Bezüglich der zweiten Fragen, die sich mit der Gesamtlaufzeit des Lokale-Suche-Algorithmus beschäftigt, untersuchen wir eine große Klasse von gewichteten Optimierungsproblemen, bei denen jede gültige Lösung als eine Teilmenge eines polynomial-großen Universums aufgefasst werden kann. Als lokale Nachbarschaft einer solchen Lösung $S$ betrachten wir alle anderen Lösungen deren symmetrische Differenz zu $S$ eine Größe von maximal $k$ hat. Wir präsentieren eine Dichotomie bezüglich $k$, ob eine lokal optimale Lösung in polynomieller Zeit gefunden

werden kann. Konkret zeigen wir, dass für $k \leq 2$, lokal optimale Lösungen in polynomieller Zeit gefunden werden können, wobei das Finden lokal optimaler Lösungen in polynomieller Zeit unwahrscheinlich erscheint für jedes $k \geq 3$.

# Contents

# Chapter 1

# Introduction

Many important real-world optimization problems are NP-hard. Hence, finding an optimal solution in polynomial time for these problems is unlikely. Consider for example Map Labeling. In this NP-hard [40] optimization problem, we are given a geographical map as well as a collection of labels for depicted parts of the map, and the goal is to find a subset $S$ of these labels to display, such that no two labels in $S$ overlap. Here, based on the fraction of the map, some labels are more important to be displayed than others. For example, in a map of Germany visualized by a navigation software on a computer screen, it seems kind of needless to place labels for small village, but placing labels for large cities as Berlin or Munich is more important. For a map of Europe, placing labels for cities seems less important than placing labels for the depicted countries. For a map showing the $100 \times 100$ m$^2$ area with your current location as center, the label of the city or even the country your are currently in are rather unimportant and it is more relevant to place labels of the surrounding streets, rivers, and sights. In Map Labeling, this is usually modeled by a weight function on the labels, and we aim to maximize to sum of weights of all placed labels. Due to the frequent usage of navigation software, specifically, navigation software where we can zoom in and out in no-time, a good set of conflict-free labels should ideally also be displayed immediately. To achieve this goal, one does not want to rely on exact algorithms which have superpolynomial worst-case running time due to the NP-hardness of the problem. Hence, in real-world applications, one has to rely on heuristic algorithms, that is, algorithms that find good but not necessarily optimal solutions quickly.

There are different concepts of heuristic algorithms, for example approximation algorithms and meta-heuristics. In *approximation algorithms* the goal is to output a solution for which the value of the objective function differs only by a (mostly)

constant factor from the value of the objective function of an optimal solution [166]. Approximation algorithms with small approximation factors do not exists for all optimization problems and if such algorithms exist, then they are mostly tailored to the optimization problem at hand. In contrast, *meta-heuristics* are high-level strategies that can be applied to a wide range of optimization problems but often give no guarantee on the quality of the outputted solution [136, 167]. Popular meta-heuristics are for example genetic algorithms [165], simulated annealing [146], ant colony algorithms [42], or local search [90]. Based on the huge success of local search [27, 90, 116], the goal of this work is to better understand the problems that occur during local search and to develop efficient algorithms solving these problems. Among other optimization problems, we consider the following generalization of Map Labeling.

> Weighted Independent Set
> **Input:** A vertex-weighted graph $G$.
> **Task:** Find an *independent set* of maximum total weight in $G$, that is, a vertex set $S$ such that no two vertices of $S$ are adjacent in $G$.

To see that Weighted Independent Set is a generalization of Map Labeling, consider an instance of Map Labeling where $L$ is the set of potential labels. Let $G$ be the vertex-weighted graph on the vertex set $L$, such that two labels $\ell_1$ and $\ell_2$ are adjacent in $G$ if and only if both labels overlap if displayed simultaneously.[1] Then, the independent sets in $G$ are exactly the label sets that can be displayed without overlaps. Hence, finding an optimal weighted independent in $G$ yields an optimal set of labels to display.

## 1.1 Hill-Climbing Local Search

In this work, we consider questions on the complexity of problems that occur during hill-climbing local search algorithms for some classical optimization problems. *Local search* algorithms [90] work by starting from an initial solution and iteratively improving it by moving to a neighboring (that is, structurally similar) solution that is better in terms of the value of the objective function. This process continues until a stopping criterion is met, such as reaching a maximum number of iterations or exceeding a given time limit. A basic hill-climbing local search algorithm takes as input an instance $I$ of a given optimization and a *local neighborhood*, that is, a function that maps each solution $S$ for $I$ to a collection of other solutions of $I$ (the *local*

[1] Recall that the weight of a label represents the importance of this label.

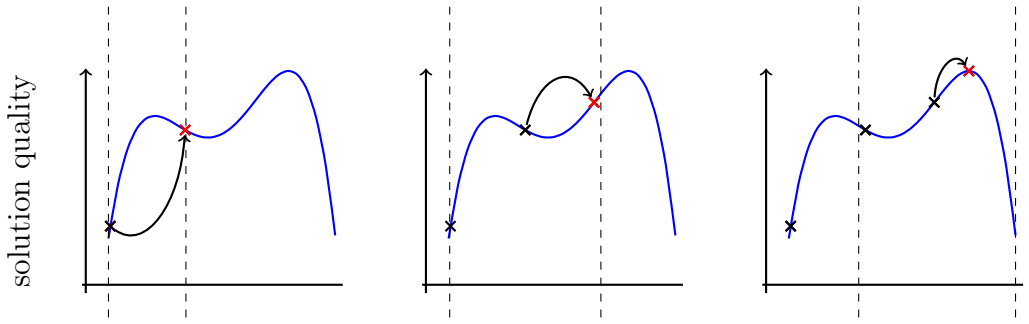



**Figure 1.1:** Three improvement steps of a hill-climbing algorithm on an abstract solution space. The first solution (bottom left) is the initial solution computed by Step 1 and the final solution (top right) is the locally optimal solution outputted by Step 3. In each iteration, the area limited by the two dashed lines indicates the local neighborhood of the current solution. In this example, the outputted solution is also globally optimal. Note that this is not always the case.

*neighborhood of* $S$) that are somewhat similar to $S$. For example for Weighted Independent Set, a natural local neighborhood of a solution $S$ are all other independent sets for which the symmetric difference to $S$ has size at most 3, the so called 3-*swap neighborhood*.

For an instance $I$ of an optimization problem and a pre-defined local neighborhood, the basic hill-climbing local search algorithm works in three steps.

Step 1: Compute some initial solution $S$ for $I$.

Step 2: If there is better solution $S'$ in the local neighborhood of the current solution $S$, replace $S$ by $S'$ and repeat Step 2.

Step 3: Output the current solution.

The outputted solution $S$ is *locally optimal* with respect to the considered local neighborhood, since no local neighbor of $S$ is better than $S$. An example for the improvement steps of such an algorithm is depicted in Figure 1.1.

Even though a locally optimal solution might be arbitrarily bad in comparison to a globally optimal solution, local search approaches very often find good solutions in practice [5, 36, 98].

Considering such a hill-climbing algorithm, several questions arise.

**Question 1.** *Which local neighborhood should we consider and how fast can we search for an improving solution in that local neighborhood?*

**Question 2.** *How fast will the algorithm output a locally optimal solution and how good is this locally optimal solution?*

Note that these two questions highly depend on the choice for the local neighborhood. Independent from the local search process, the following question arises.

**Question 3.** *What is a good starting solution and how fast can we compute it?*

In this work, we do not consider Question 3 because for each problem considered in this work, feasible starting solutions can be found in linear time. Since this work aims at better understanding the difficulties that need to be overcome by local search algorithms, Question 3 is thus not interesting from a theoretical viewpoint for the considered problems and should rather be evaluated experimentally.

## 1.2 Variations of Basic Hill-Climbing

There are many variations and extensions of the basic hill-climbing algorithm [90]. In this work, we consider (deterministic) exhaustive hill-climbing local search for single criteria optimization problems. Below, we list some of the most prominent and successful practically applied variations to clarify the setting we consider in this work and put it in a larger context.

**Exhaustive search vs. random search.** The basic hill-climbing algorithm performs so called *exhaustive search*, since the algorithm only terminates, when the current solution has no better solution in the local neighborhood. Since the local neighborhood of a given solution $S$ might be very large, exhaustive search may be too time-consuming and instead, one might be interested in outputting the current solution even if it is not locally optimal. This can be done for example by choosing a local neighbor $S'$ of the current solution $S$ randomly and checking whether $S'$ is better than $S$. If this procedure does not find a better solution after $r$ consecutive random choices, the current solution $S$ is outputted (even though $S$ might not be locally optimal). In contrast to exhaustive search, such an approach is called *random search* [90].

**Strict local search vs. permissive local search.** A different way to adapt the basic hill-climbing algorithm is to omit the limitation that the current solution $S$ can only be replaced by a better solution in the local neighborhood of $S$. The basic hill-climbing algorithm is *strict* in the sense that we only want to replace the current

solution $S$ by a better solution in the local neighborhood of $S$. Instead, a hill-climbing algorithm in which we ask for any better solution to replace the current solution $S$ is called *permissive* [69]. Still, such a *permissive* hill-climbing algorithm is only allowed to output locally optimal solutions. Depending on the structure of the solution $S$, finding any better solution may be easier than performing an exhaustive search over the whole local neighborhood of $S$ and might also help escaping local optima [69]. The latter is the case, since such a permissive hill-climbing algorithm is allowed to (but not forced to) stop, when it finds a locally optimal solution.

**Variable neighborhood search (VNS).** Instead of only considering a single local neighborhood, one can also consider multiple local neighborhoods of a given solution. The local neighborhood for which one tries to find a better solution can be chosen according to some priority ordering of the local neighborhoods (that may change over time) or randomly. In the first case, the eventually outputted solution then guarantees to be locally optimal for all considered local neighborhoods. The idea is that if the current solution is locally optimal with respect to one local neighborhood, then a different local neighborhood may still contain improving solutions. Hence, the different local neighborhoods may help each other out of bad local optima. Ideally, this will then lead to better solutions than the ones one might find by considering these local neighborhoods only individually. Such an approach with more than one local neighborhood is called *variable neighborhood search* [126].

For example for Weighted Independent Set, one could additionally consider a change-neighborhood where one is allowed to add a single vertex to the current solution and afterwards remove all its neighbors from the solution, to again obtain an independent set. On the one hand, since a vertex may have an arbitrarily large degree in the current solution, this local neighborhood contains solutions that are not contained in the 3-swap neighborhood. On the other hand, since this change-neighborhood only allows to add a single vertex to the solution, the 3-swap neighborhood contains solutions that are not contained in the change-neighborhood. Hence, no local neighborhood contains the other and any of these local neighborhoods may help finding better solutions when the current solution is locally optimal with respect to the other local neighborhood.

**Iterated local search (ILS).** A different approach to find good local optima is to run the basic hill-climbing algorithm multiple times on different starting solutions and to output the best found solution over all such runs. *Iterated local search* [118, 158] is one such approach, where instead of outputting the found locally optimal solution $S$ by the basic hill-climbing algorithm, one obtains a new solution $S'$ that

shares some structure with $S$ by performing perturbations on $S$, and afterwards applies the basic hill-climbing algorithm again with $S'$ as the new starting solution. This is usually repeated a given number of times or until a time limit is exceeded. Afterwards, the best overall found solution is outputted. The perturbation of the found locally optimal solution $S$ is mostly done randomly [158] and deciding which and how many perturbations are applied is a difficult task. On the one hand, the fewer perturbations are applied, the more likely it is that the new run of the basic hill-climbing algorithm finds the locally optimal solution $S$ again. On the other hand, the more perturbations are applied, the more structure of the solution $S$ is lost, which makes it more likely, that the new solution $S'$ is way worse than $S$.

To prevent considering previously encountered solutions, one may additionally use so called *tabu lists* to store previously encountered solutions. When searching for a better solution in the local neighborhood, the solutions that are contained in the tabu list are excluded from the set of possible better solutions. Hence, the algorithm will not consider any solution from the tabu list again. This sub-variation of ILS is called *Tabu Search* [113].

## 1.3 Scope of this Work

We consider for several local search problems Questions 1 and 2 with respect to "scalable neighborhoods". A *scalable neighborhood* is a local neighborhood which is defined over a distance function $d$ between the solutions of the problem instance at hand. More precisely, for a distance function $d$ and a non-negative integer $k$, the *$k$-neighborhood* of a solution $S$ with respect to $d$ is the collection of all solutions for which the distance from $S$ with respect to $d$ is at most $k$. That is, the $k$-neighborhood of $S$ is the collection of all solutions $S'$ for which $d(S, S') \leq k$ holds. In most implementations of local search algorithms, one considers the scalable $k$-neighborhood for small constant values of $k$ [98,99,120]. Note that the use of scalable neighborhoods can also be considered as a variable neighborhood search: When the hill-climbing algorithm finds a locally optimal solution with respect to the $k$-neighborhood, instead of outputting this solution, increase the value of $k$ by 1 and continue the hill-climbing algorithm. Hence, the $(k+1)$-neighborhood might allow to escape the previous local optima.

To analyze the two considered questions for local search algorithms, the most established frameworks are: (i) analyzing Question 1 from the perspective of parameterized complexity for local search problems that use scalable neighborhoods and (ii) analyzing Question 2 with respect to the complexity class for local search problems: PLS.

### 1.3.1 Related Work

The study on the parameterized complexity of local search problems (often referred to as *parameterized local search*) that use scalable neighborhoods was first initiated by Marx [122] with respect to Traveling Salesman Problem and the scalable neighborhood based on the symmetric difference between the edge sets of the solutions. Since in most practical applications the $k$-neighborhood is only considered for small constant values and the value of $k$ is independent from the actual problem instance at hand, one would ideally obtain an algorithm that searches the $k$-neighborhood in $f(k) \cdot |I|^{\mathcal{O}(1)}$ time. Marx [122], however, showed that searching this $k$-neighborhood is W[1]-hard when parameterized by the search radius $k$, that is, there is presumably no algorithm running in $f(k) \cdot |I|^{\mathcal{O}(1)}$ time that determines whether the current solution is locally optimal with respect to the $k$-neighborhood. Since then, the parameterized complexity of parameterized local search was analyzed for many optimization problems. With few exceptions, all considered parameterized local search problems share the same parameterized intractability result, namely W[1]-hardness when parametrized by the search radius $k$ [23, 43, 52, 68, 69, 77, 82, 83, 87, 98, 122, 135, 161, 164]. Besides that, for many considered parameterized local search problems, FPT-algorithms were developed with respect to parameter combinations including the search radius $k$ and other structural parameters. Parameterized local search was mostly analyzed from a theoretical point of view, but the few experimental evaluations showed that hill-climbing based on the developed FPT-algorithms perform well either as standalone algorithms or as post-processing algorithms for other state-of-the-art heuristics [68, 77, 87, 98].

To analyze Question 2, Johnson et al. [95] introduced the complexity class PLS for local search problems. Roughly speaking, PLS contains those local search problems for which each improvement step takes only polynomial time.[2] Hence, intuitively, the worst-case running time for finding a locally optimal solution for problems in PLS is upper-bounded by the maximal number of improvement steps of a respective hill-climbing algorithm times some polynomial factor. So far, no polynomial-time algorithm is known that finds a locally optimal solution for any PLS-hard problem. Since the introduction of PLS in 1988, many local search problems where shown to be PLS-hard [46, 47, 95, 125, 137, 151, 154].

[2] A formal definition of PLS is given in Section 2.7.

### 1.3.2 Our Results

In Chapters 3 to 6, we analyze the parameterized complexity of determining whether there is a better solution in the (scalable) local neighborhood. We consider four classical optimization problems on graphs and one problem on inferring evolutionary trees with respect to parameter combinations including the search radius $k$ and other structural parameters. For all these problems, we present conditional running time lower bounds of the form $|I|^{\Omega(k)}$, where $|I|$ is the whole input size and $k$ is the search radius of the scalable neighborhood. Except for one considered local search problem (namely Cluster Editing), we also extend the conditional running time lower bounds to the permissive version of the respective problem, that is, the version where we are allowed to return *any* better solution. For each considered local search problem, we then provide algorithms that solve the corresponding problem in $\ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time, where $\ell$ is some problem-specific parameter fulfilling $\ell \leq |I|$. For one of the considered optimization problems, namely Max $c$-Cut, we implemented and engineered the developed algorithm and experimentally evaluated its performance as post-processing for a state-of-the-art heuristic.

In Chapter 7, we analyze the worst-case running time of finding locally optimal solutions for a large class of weighted optimization problems, where each solution is a subset of some given universe. We consider the most natural scalable neighborhood for these problems, namely the collection of all solutions for which the symmetric difference with the current solution has size at most $k$. With respect to this scalable neighborhood, we show that one can find for many such weighted optimization problems a locally optimal solution in polynomial time when considering only the 2-neighborhood, but for the 3-neighborhood, the considered problems are PLS-hard. One of the optimization problems for which we derive this dichotomy is Weighted Independent Set.

# Chapter 2

# Preliminaries

In this chapter, we introduce some basic concepts of graph theory, (parameterized) complexity theory, and local search problems that we will use throughout this work. Additionally, in Section 2.8, we present a generic algorithm that helps us obtain efficient algorithms for the local search problems considered in Chapters 4 and 5. Furthermore, this algorithm also improves on the running time of previously best algorithms for other previously considered local search problems.

## 2.1 Set Notation

For integers $i$ and $j$ with $i \leq j$, we define $[i, j] := \{k \in \mathbb{N} \mid i \leq k \leq j\}$. For a set $A$ and an integer $k$, we denote with $\binom{A}{k}$ the collection of all size-$k$ subsets of $A$. For two sets $A$ and $B$, we denote with $A \oplus B := (A \setminus B) \cup (B \setminus A)$ the *symmetric difference* of $A$ and $B$.

A *partition* $\mathcal{P}$ of $A$ is a collection of non-empty and pairwise disjoint subsets of $A$ such that $\cup_{P \in \mathcal{P}} P = A$. Let $r \geq 2$. An $r$-partition of a set $A$ is an $r$-tuple $(B_1, \ldots, B_r)$ of (potentially empty) subsets of $A$, such that each element of $A$ is contained in exactly one set of $(B_1, \ldots, B_r)$. Intuitively, an $r$-partition of $A$ is a permutation of the subsets contained in a partition $\mathcal{P}$ of $A$ of size at most $\ell \leq r$ plus $\ell - r$ empty sets. For $r = 2$, we may call a 2-partition $(A, B)$ simply a *partition*.

For some set $A$, we call a function $\chi \colon A \to \mathbb{N} \setminus \{0\}$ a *coloring of* $A$. If for some $c \in \mathbb{N}$, no element of $A$ is assigned a value larger than $c$ by $\chi$, we call $\chi$ a *$c$-coloring of* $A$. Note that there is a one-to-one correspondence between the $r$-partitions of a set $A$ and the $r$-colorings of $A$. For a function $f \colon A \to B$ and $C \subseteq A$ we denote by $f|_C$ the function $f$ restricted to $C$. Let $f \colon A \to B$ and $g \colon A \to B$ be functions and let $C \subseteq A$. Then, we say that $f$ and $g$ *agree* on $C$, if $f|_C = g|_C$. A

function $d : A \times A \to \mathbb{N}$ is a *distance function* if for each two elements $a$ and $b$ of $A$, $d(a, b) = 0$ if and only if $a = b$.

## 2.2 Graph Theory Notation

In this section, we focus on graph notation. For a more detailed introduction to concepts of graph theory, we refer to the standard monographs [38, 173]. An (undirected) graph $G = (V, E)$ consists of a set of *vertices* $V$ and a set of *edges* $E \subseteq \binom{V}{2}$. We also denote the vertex and edge set of a graph $G$ by $V(G)$ and $E(G)$, respectively. Let $u$ and $v$ be distinct vertices of $G$ with $\{u, v\} \notin E$, then we call $\{u, v\}$ a *non-edge* of $G$. The *complement graph* of $G$ is the graph $(V, \binom{V}{2} \setminus E)$. For vertex sets $S \subseteq V$ and $T \subseteq V$, we denote with $E_G(S, T) := \{\{s, t\} \in E \mid s \in S, t \in T\}$ the edges between $S$ and $T$ and we use $E_G(S) := E_G(S, S)$ as a shorthand. We define $G[S] := (S, E_G(S, S))$ as the *subgraph of $G$ induced by $S$*. For a vertex $v \in V$, we denote with $N_G(v) := \{w \in V \mid \{v, w\} \in E\}$ the *open neighborhood* of $v$ in $G$ and with $N_G[v] := \{v\} \cup N_G(v)$ the *closed neighborhood* of $v$ in $G$. Analogously, for a vertex set $S \subseteq V$, we define $N_G[S] := \bigcup_{v \in S} N_G[v]$ and $N_G(S) := N_G[S] \setminus S$. The *degree of a vertex $v$ in $G$* is the number of neighbors of $v$ in $G$, that is, $|N_G(v)|$. We denote the *maximum degree of $G$*, that is, the largest degree of any vertex of $G$, by $\Delta(G)$. If $G$ is clear from the context, we may omit the subscript.

A *path* $P$ in a graph $G$ is a sequence of pairwise distinct vertices of $G$ that are consecutively adjacent. The length of a path $P := (v_0, \ldots, v_r)$ is the number of edges of $P$, that is, $r$. We refer to the first and the last vertex of $P$ as the *terminals of $P$*. A graph $G$ is *connected* if there is a path between any pair of vertices of $V$. A vertex set $S \subseteq V$ is *connected* if $G[S]$ is a connected graph. If $S$ is inclusion-maximal under this property, we call $S$ a *connected component of $G$*. Let $v$ and $w$ be two vertices of $G$. We say that *$v$ and $w$ have distance $i$* if the length of a shortest path between $v$ and $w$ is $i$. If there is no path between $v$ and $w$, that is, if $v$ and $w$ are not part of the same connected component, we say that *$v$ and $w$ have distance $\infty$*. The *diameter* of $G$ is the maximum distance between any two vertices of $G$.

A vertex set $S \subseteq V$ is a *vertex cover* of $G$ if at least one endpoint of each edge in $E$ is contained in $S$. Similar, a vertex set $S \subseteq V$ is an *independent set* in $G$ if $V \setminus S$ is a vertex cover of $G$, that is, the vertices of $S$ are pairwise non-adjacent. A vertex set $S \subseteq V$ is a *dominating set* of $G$ if for each vertex $v$ of $G$, $S$ contains $v$ or at least one neighbor of $v$. An edge set $M \subseteq E$ is a *matching* in $G$ if no two edges of $M$ share an endpoint. A matching $M$ is *perfect* if each vertex of $G$ is contained in exactly one edge of $M$.

The *subdivision of an edge $e \in E$* in $G$ results in the graph $G'$ obtained by

removing $e$ from $G$ and adding a new vertex which is adjacent to both endpoints of $e$. Let $v$ be a vertex of degree 2 in $G$. The *suppression of* $v$ in $G$ results in the graph $G'$ obtained by removing $v$ from $G$ and joining both neighbors of $v$ by an edge.

For some integer $c \geq 2$, a graph $G$ is *$c$-partite* if there is a $c$-partition $(V_1, \ldots, V_c)$ of $V$ such that $V_i$ is an independent set in $G$ for each $i \in [1, c]$. We call a 2-partite graph *bipartite*. If a graph $G$ is $c$-partite and we are given a $c$-partition $(V_1, \ldots, V_c)$ of $V$, we may implicitly assume that $V_i$ is an independent set in $G$ for each $i \in [1, c]$. A graph $G$ is *$r$-regular* for some integer $r$ if each vertex of $G$ has exactly $r$ neighbors in $G$. The *$h$-index* of a graph $G$, denoted by $h(G)$, is the largest integer such that $G$ has at least $h(G)$ vertices of degree at least $h(G)$. The *degeneracy of* $G$ is the smallest integer $d(G)$, such that each subgraph of $G$ contains at least one vertex of degree at most $d$. A *degeneracy ordering of* $G$ is an ordering $\pi$ of the vertices of $G$, such that for each $i \in [1, |V|]$ the vertex at position $i$ in $\pi$ has at most $d$ neighbors in $G$ with larger positions in $\pi$. For each graph, such a degeneracy ordering exists.

## 2.3 Computational Complexity

In this section, we summarize the main tools we use throughout this work to analyze the computational complexity of the considered problems [10, 65, 138].

Formally, a *decision problem* $L$ is a subset of $\{0,1\}^*$. Informally, the task of a decision problem is to decide whether a given word $x \in \{0,1\}^*$ is contained in $L$. We say that $x$ is a *yes-instance of* $L$ if $x \in L$, and a *no-instance of* $L$, otherwise. Each decision problems $L$ considered in this work is decidable, that is, there is an algorithm that terminates and that decides whether a given word $x$ is a yes-instance of $L$.

A *complexity class* $\mathcal{L}$ is a collection of decision problems. The two arguably most prominent and important complexity classes are P and NP. Here, P is the complexity class containing all decision problems that can be solved by a deterministic Turing machine in polynomial time. That is, a decision problem $L$ is contained in P if and only if there is an algorithm $A$ that decides deterministically in $|x|^{\mathcal{O}(1)}$ time, whether a given word $x$ is contained in $L$. Similarly, NP is the complexity class containing all decision problems that can be solved by a non-deterministic Turing machine in polynomial time. Note that $\mathrm{P} \subseteq \mathrm{NP}$. While it remains open whether this inclusion is proper, it is widely believed that $\mathrm{P} \neq \mathrm{NP}$.

A decision problem $L'$ is NP-*hard* if for each problem $L$ in NP, there is a polynomial-time reduction from $L$ to $L'$. Here, a *polynomial-time reduction from $L$ to $L'$* (denoted by $L \leq_P L'$) is an algorithm that takes a word $x$ and computes a word $x'$ in $|x|^{\mathcal{O}(1)}$ time, such that $x$ is a yes-instance of $L$ if and only if $x'$ is a

yes-instance of $L'$. Note that the composition of two polynomial-time reductions is a polynomial-time reduction too. Hence, to show that a problem $L$ is NP-hard, it suffices to show that there is an NP-hard problem $L'$, such that $L' \leq_P L$. A decision problem $L$ is NP-*complete* if $L$ is NP-hard and contained in NP. Intuitively, an NP-complete problem $L$ is as hard as any problem in NP with respect to polynomial-time solvability. This is due to the fact that each problem $L'$ of NP can be solved by reducing it to $L$ in polynomial time and afterwards solving the corresponding instance of $L$.

An example for an NP-complete problem is 3-SAT, where the input is a Boolean formula $F$ in 3-CNF and one has to decide whether there is an assignment of the variables of $F$ that satisfies all clauses of $F$.

It is further widely assumed that some NP-hard problems cannot be solved in $2^{o(|I|)}$ time. For example, the Exponential Time Hypothesis (ETH) postulates that there is a constant $c > 0$, such that 3-SAT cannot be solved in $2^{c \cdot n} \cdot n^{\mathcal{O}(1)}$ time [93], where $n$ denotes the number of variables of the input formula $F$. Assuming the ETH, this implies that 3-SAT cannot be solved in $2^{o(n)} \cdot n^{\mathcal{O}(1)}$ time. This has further implication also to other decision problems. For example, it was shown that the following problem cannot be solved in $f(k) \cdot |I|^{o(k)}$ time for any computable function $f$, unless the ETH fails [35].

Clique
**Input:** A graph $G = (V, E)$ and $k \in \mathbb{N}$.
**Question:** Is there a clique of size at least $k$ in $G$?

The same intractability result was also shown for the more restricted Multicolored Clique [35].

Multicolored Clique
**Input:** A graph $G = (V, E)$ and $k \in \mathbb{N}$, such that $G$ is $k$-partite.
**Question:** Is there a clique of size at least $k$ in $G$?

This holds even if a $k$-partition of $G$ is provided [35]. When reducing from Multicolored Clique, we may thus implicitly assume that a $k$-partition of $G$ is provided.

## 2.4 Parameterized Complexity

In *parameterized complexity theory*, the goal is to find efficient algorithms for NP-hard problems under the assumption that specific input-parameters are small. In

the following, we provide the definitions of the relevant aspects of parameterized complexity theory that are relevant to this work. For a more detailed overview on parameterized complexity theory, we refer to the standard monographs [35, 44, 131].

A *parameterized problem* $L$ is a subset of $\{0,1\}^* \times \mathbb{N}$. Intuitively, a parameterized problem consists of a decision problem which is equipped with an additional *parameter*. In general, the goal for parameterized problems is to analyze whether the following most desirable running time is possible.

**Definition 2.1.** A parameterized problem $L$ is *fixed-parameter tractable* if there is a computable function $f$ such that for every instance $(x,k) \in \{0,1\}^* \times \mathbb{N}$ it can be decided in $f(k) \cdot |x|^{\mathcal{O}(1)}$ time whether $(x,k) \in L$.

The parameterized complexity class FPT contains exactly the parameterized problems that are fixed-parameter tractable. Moreover, we call a running time of $f(k) \cdot |x|^{\mathcal{O}(1)}$ *FPT-time for* $k$.

**Definition 2.2.** A parameterized problem $L$ is *slicewise polynomial* if there is a computable function $f$ such that for every instance $(x,k) \in \{0,1\}^* \times \mathbb{N}$ it can be decided in $|x|^{f(k)}$ time whether $(x,k) \in L$.

The class XP contains exactly the parameterized problems that are slicewise polynomial. Clearly, each problem in FPT is also contained in XP and it is widely assumed that this inclusion is proper. Both parameterized complexity classes are designed to capture "efficiently" solvable problems. The difference between FPT and XP is, that the degree of this polynomial function is independent of $k$ in case of FPT, but might depend on $k$ in case of XP.

To find evidence that a parameterized problem is not contained in FPT, Downey and Fellows [44] introduced a hierarchy of parameterized complexity classes which all are supersets of FPT: For each $i \geq 1$ they introduced the parameterized complexity class W[$i$].

These complexity classes form the W-*hierarchy* and is widely believed that the inclusions FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots \subseteq$ XP are proper. Similar to NP-hardness, Downey and Fellows [44] introduced the notion of parameterized problems that are hard for some class W[$i$]. A parameterized problem $L$ is W[$i$]-*hard* if there is a parameterized reduction from each problem of W[$i$] to $L$. Here, a parameterized reduction is formally defined as follows.

**Definition 2.3.** A *parameterized reduction* from a parameterized problem $L$ to a parameterized problem $L'$ is an algorithm that transforms an instance $I = (x,k)$ of $L$ into an instance $I' = (x',k')$ of $L'$ and runs in $f(k) \cdot |x|^{\mathcal{O}(1)}$ time such that (i) $I \in L$

if and only if $I' \in L'$ and (ii) $k' \leq g(k)$ for some computable functions $f$ and $g$. If $k' = k$ for all instances of $L$, the reduction is *parameter-preserving*.

Observe that $k'$ depends only on $k$ and that the running time of a parameterized reduction is exactly FPT-time. This implies that if a W[$i$]-hard parameterized problem $L$ is contained in FPT, then each problem of W[$i$] is contained in FPT, that is, FPT = W[$i$]. Since it is widely believed that FPT is a proper subset of W[1], this implies that W[1]-hardness of a parameterized problem $L$ gives evidence that $L$ is presumably not contained in FPT.

A known W[1]-hard problem is Independent Set parameterized by the size of the sought solution $k$ [44].

Independent Set
**Input:** A graph $G = (V, E)$ and a positive integer $k$.
**Question:** Is there an independent set of size at least $k$ in $G$?

In contrast, the dual problem Vertex Cover is in FPT when parameterized by the size $k$ of the sought solution [44].

Vertex Cover
**Input:** A graph $G = (V, E)$ and a positive integer $k$.
**Question:** Is there a vertex cover of size at most $k$ in $G$?

Moreover, recall that Clique and Multicolored Clique cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function, unless the ETH fails [35]. Note that this excludes an FPT-algorithm for Clique and Multicolored Clique when parameterized by $k$. Hence, also the ETH can give evidence that some parameterized problem do not admit FPT-algorithms [35].

## 2.5 Optimization and Local Search Problems

Next, we formally define local search problems [90, 139]. To this end, we first have to formally define optimization problems.

**Definition 2.4.** An *optimization problem* $L$ is specified to be either a *minimization* or a *maximization* problem and consists of

- a set $\mathcal{D}_L \subseteq \{0,1\}^*$ of *instances*,
- for each instance $I \in \mathcal{D}_L$, a set of *feasible solutions* $\mathcal{S}_L(I) \subseteq \{0,1\}^*$ for $I$, and

- an *objective function* $\text{val}_L$ which assigns a rational number to each pair $(I, S)$, where $I$ is an instance of $L$ and $S$ is a feasible solution for $I$.

An optimization problem $L$ is an NP-*optimization problem* if (i) the encoding length of each solution $S \in \mathcal{S}_L(I)$ of $I$ is polynomially bounded by $|I|$, (ii) one can determine in polynomial time for each pair $(I, S)$ whether $S \in \mathcal{S}_L(I)$, and (iii) the objective function can be evaluated in polynomial time.

Let $I$ be an instance of an optimization problem $L$ and let $S$ and $S'$ be feasible solutions for $I$. We say that $S$ is *better than* or *improving over* $S'$ if (i) $L$ is a maximization problem and $\text{val}_L(I, S) > \text{val}_L(I, S')$ or (ii) $L$ is a minimization problem and $\text{val}_L(I, S) < \text{val}_L(I, S')$.

An example of an NP-optimization maximization problem is Max Cut.

Max Cut
**Input:** A graph $G = (V, E)$ and an edge-weight function $\omega \colon E \to \mathbb{Z}$.
**Task:** Find a partition $(A, B)$ of $V$ that maximizes $\sum_{e \in E(A,B)} \omega(e)$.

For an instance $I := (G = (V, E), \omega)$ of Max Cut, the set of feasible solutions are exactly the partitions of $V$ and for each partition $(A, B)$ of $V$, the objective functions assigns value $\sum_{e \in E(A,B)} \omega(e)$ to $(I, (A, B))$.

Another example of an NP-optimization maximization problem is Max Sat.

Max Sat
**Input:** A Boolean formula $F$ in CNF.
**Task:** Find an assignment of the variables of $F$ that satisfies a maximum number of clauses of $F$.

We are now ready to formally define the main kind of problems we analyze in this work: local search problems.

**Definition 2.5.** A *local search problem* $(L, \mathcal{N})$ consists of

- an optimization problem $L$ and
- a *neighborhood structure* $\mathcal{N}$ *for* $L$ that maps for each instance $I$ of $L$, each valid solution $S$ of $I$ to a set $\mathcal{N}(I, S) \subseteq \mathcal{S}_L(I)$ of valid solutions for $I$, the *neighbors of* $S$ *with respect to* $\mathcal{N}$.

For a given instance $I$ of $L$ a *locally optimal solution* $S$ *with respect to* $\mathcal{N}$ is a feasible solution $S$ for $I$ such that no solution in $\mathcal{N}(I, S)$ is better than $S$. We may write a local search problem $(L, \mathcal{N})$ as $L/\mathcal{N}$.

An example for a local search problem is Max Cut/flip, where the neighbors of a partition $(A, B)$ are all partitions obtainable by moving a single vertex from $A$ to $B$, or vice versa. This neighborhood may be formally defined as follows: A *flip of a vertex $v \in V$ in a partition* $(A, B)$ is the partition $(A', B')$, where $A' := A \oplus \{v\}$ and $B' := B \oplus \{v\}$. Moreover, we say that $(A', B')$ *is improving over* $(A, B)$ if the total weight of the cut $E_G(A', B')$ is larger than the total weight of the cut $E_G(A, B)$. Furthermore, we say that a partition $(A, B)$ is *flip-optimal* if there is no vertex $v \in V$ such that the flip $(A', B')$ of $v$ in $(A, B)$ is improving over $(A, B)$.

Recall that we aim to analyze the questions on (i) how fast a better solution in the local neighborhood can be found, if one exists, and (ii) how fast a locally optimal solution can be found. In Section 2.6 we will formally introduce the kind of problems we analyze with respect to this first question, and in Section 2.7, we will introduce the framework to analyze the second question.

## 2.6 Parameterized Local Search

In most practical applications, the considered local neighborhoods are usually defined over a simple operation, like the above-mentioned flip of a single vertex for Max Cut. Since such simple operations are rather restricted and thus might lead to bad locally optimal solutions, one may allow larger neighborhoods that contain all solutions that can be obtained by performing $k$ such simple operations consecutively or simultaneously. In this work, we consider scalable neighborhoods which generalize this concept of applying multiple simple operations consecutively (or simultaneously) to obtain a neighboring solution.

**Definition 2.6.** Let $L$ be an optimization problem and let $I$ be a instance of $L$. Moreover let $d$ be a distance measure between the solutions of $I$. For each solution $S$ of $I$ and each integer $k$, the *(scalable) $k$-neighborhood of $S$ with respect to $d$* consists of all solutions of $I$ that have distance at most $k$ to $S$.

An example for such a scalable neighborhood is the *$k$-flip neighborhood* [66, 102, 161] for partition and coloring problems. This scalable neighborhood is a direct generalization of the flip neighborhood:

**Definition 2.7.** Let $X$ be a set and let $\chi$ and $\chi'$ be colorings of $X$. The *flip* between $\chi$ and $\chi'$ is defined as $D_{\text{flip}}(\chi, \chi') := \{x \in X \mid \chi(x) \neq \chi'(x)\}$ and the *flip distance* between $\chi$ and $\chi'$ is defined as $d_{\text{flip}}(\chi, \chi') := |D_{\text{flip}}(\chi, \chi')|$.

We say that $\chi$ and $\chi'$ are *$k$-(flip-)neighbors* if $d_{\text{flip}}(\chi, \chi') \leq k$. Hence, the scalable $k$-neighborhood of a coloring $\chi$ with respect to $d_{\text{flip}}$ consists of all colorings that

can be obtained by flipping the color of $k$ elements of $X$ consecutively or simultaneously. In other words, the scalable $k$-neighborhood with respect to $d_{\text{flip}}$ is defined over the simple operation of flipping the color of a single element at a time. Still, the definition of scalable neighborhoods also allows for distance measures that are not defined over simple operations. An example of a scalable neighborhood over such a distance measure is considered in Chapter 6, where we study local search for the Maximum Parsimony problem.

Based on the definition of scalable neighborhoods, we are now ready to formally define the computational problems we consider in this work from a parameterized complexity point of view. Let $L$ be an optimization problem and let $d$ be a distance measure between the solutions of the instances of $L$. In *parameterized local search* for $L$ and $d$, one is given an instance $I$ of $L$, a feasible solution $S$ for $I$, and an integer $k$, and one has to determine whether the scalable $k$-neighborhood with respect to $d$ contains a better solution than $S$. This is defined as follows.

$d$-LS $L$
**Input:** An instance $I$ of $L$, a solution $S$ for $I$, and an integer $k$.
**Question:** Is there a better solution $S'$ for $I$ with $d(S, S') \leq k$?

We refer to $k$ as the *search radius*. This is motivated by the fact that for most NP-optimization problems the scalable $k$-neighborhood with respect to the most natural distance measures $d$ have size $|I|^{\mathcal{O}(k)}$.

Marx [122] was the first to analyze $d$-LS $L$ with $L$ being the famous Traveling Salesman Problem and $d$ being the distance measure defined over the symmetric difference between the edge sets of the solutions, that is, the distance measure over which the so called $k$-OPT-neighborhood is defined. Marx [122] showed that $d$-LS $L$ is W[1]-hard when parameterized by the search radius $k$. Since then, the parameterized complexity of parameterized local search was analyzed for many optimization problems and with only few exceptions [59, 77], all considered parameterized local search problems share the same parameterized intractability result, namely W[1]-hardness when parametrized by the search radius $k$ [23, 43, 52, 68, 69, 82, 83, 87, 98, 135, 161, 164]. Note that $d$-LS $L$ only allows for a better solution in the $k$-neighborhood and one is not allowed to return *any* better solution of arbitrary large distance. Gaspers et al. [69] showed that this restriction can be exploited: They showed that $d$-LS $L$ is NP-hard and W[1]-hard when parameterized by the search radius, even on instances where an optimal solution can be found in polynomial time. This result was shown for $d$-LS $L$ with $L$ being Vertex Cover and $d$ being the distance measure defined over the symmetric difference between any two solutions. Hence, since the overall goal is to develop efficient algorithms for the subroutine of

finding a better solution within a hill-climbing algorithm, only asking for a better solutions in the local neighborhood is in some situations an unnecessarily complicated restriction. This motivated the study of *permissive parameterized local search problems* [69], where we are allowed to provide *any* better solution, instead of only allowing solutions in the scalable local neighborhood. This is formally defined as follows.

PERMISSIVE $d$-LS $L$
**Input:** An instance $I$ of $L$, a solution $S$ for $I$, and an integer $k$.
**Task:** Find a better solution $S'$ for $I$, or correctly output that there is no better solution $S'$ for $I$ with $d(S, S') \leq k$.

To distinguish between $d$-LS $L$ and PERMISSIVE $d$-LS $L$, we may refer to them as the *strict* and *permissive* version of $d$-LS $L$, respectively.

We will show that most considered parameterized local search problems in this work do not admit FPT-algorithms when parameterized by their respective search radius, even when considering the permissive version of these problems. To obtain these results, we will present parameterized reductions from W[1]-hard problems to the strict version of $d$-LS $L$, such that in the constructed instance of $d$-LS $L$, the initial solution $S$ is locally optimal if and only if $S$ is globally optimal. Hence, on these constructed instances, an algorithm for the permissive version would also solve the strict version: If the permissive algorithm outputs any better solution, the current solution is not globally optimal and thus not locally optimal. Otherwise, if the permissive algorithm outputs that there is no better solution is the local neighborhood, then obviously this answer is also the answer of the strict algorithm.

If a parameterized local search problem does not admit an FPT-algorithm when parameterized by $k$, algorithms where the superpolynomial running time part depends only on the search radius $k$ are unlikely. Still, we aim to find algorithms for these problems that achieve running times which can be considered practical even though the superpolynomial running time part also depend on other parameters. We formalize the class of running time functions we aim to achieve as follows.

**Definition 2.8.** Let $f\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a function. We say that $f$ *grows mildly with respect to* $\ell$ *and strongly with respect to* $k$ if $f(\ell, k) \in \mathcal{O}(\ell^{g(k)})$ for some computable function $g$ depending only on $k$.

We are interested in obtaining algorithms whose running time grows strongly only with respect to $k$ and mildly with respect to some other parameters. In our opinion, such running times are desirable for parameterized local search problems that

are W[1]-hard when parameterized by their search radius $k$, since in most practical applications, the search radius $k$ can be considered as a small constant.

Still, the usefulness of for such running times is not limited to local search problems. Instead, it may be useful whenever

- two parameters $k$ and $\ell$ are studied,
- $k$ is known to be very small on relevant input instances,
- $k$ is known to be much smaller than $\ell$ on these instances,
- and the problem is W[1]-hard with respect to $k$.

## 2.7 The Complexity Class PLS

Regarding the second question we try to analyze in this work, namely the complexity of finding locally optimal solutions for local search problems, we consider hardness with respect to a complexity class called PLS. Johnson et al. [95] introduced the complexity class PLS for local search problems to analyze the complexity of this task.

**Definition 2.9.** A local search problem $(L, \mathcal{N})$ *is in the complexity class* PLS if

- $L$ is an NP-optimization problem,
- there is an algorithm which computes in polynomial time some feasible solution $S$ for a given instance $I$ of $L$, and
- there is an algorithm which in polynomial time determines whether a given solution $S$ is locally optimal with respect to $\mathcal{N}$ for an instance $I$ of $L$ and, if this is not the case, outputs a better neighbor of $S$.

An example for a local search problem contained in PLS is Max Cut/flip: Each possible partition of the vertex set is a feasible solution and for each partition, one can determine in polynomial time whether a given partition can be improved by flipping a singe vertex.

To categorize the hardest local search problems in PLS, Johnson et al. [95] further introduced reductions between local search problems.

**Definition 2.10.** Let $(L_1, \mathcal{N}_1)$ and $(L_2, \mathcal{N}_2)$ be local search problems. We say that $(L_1, \mathcal{N}_1)$ *is* PLS-*reducible to* $(L_2, \mathcal{N}_2)$ if for each instance $I_1$ of $L_1$, one can compute an instance $I_2$ of $L_2$ in polynomial time and define a polynomial-time computable *solution-mapper* $f$ that *preserves local optimality.*

Here, a function $f$ is a solution-mapper if $f$ maps each solution $S_2$ of $I_2$ to a solution $f(S_2)$ of $I_1$. Furthermore, $f$ preserves local optimality if for each locally optimal solution $S_2$ for $I_2$ with respect to $\mathcal{N}_2$, $f(S_2)$ is locally optimal for $I_1$ with respect to $\mathcal{N}_1$.

Similar to NP-completeness, a local search problem $(L, \mathcal{N})$ is PLS-*hard* if for each local search problem $(L', \mathcal{N}')$ in PLS, there is a PLS-reduction from $(L', \mathcal{N}')$ to $(L, \mathcal{N})$. Since the composition of two PLS-reductions is again a PLS-reduction, this can be shown by proving a PLS-reduction from any PLS-hard local search problem $(L', \mathcal{N}')$. Moreover, $(L, \mathcal{N})$ is PLS-*complete* if $(L, \mathcal{N})$ is contained in PLS and PLS-hard.

An example for a PLS-complete local search problem is Max Cut/flip [151]. The PLS-completeness of Max Cut/flip holds even on graphs of degree at most 5 [47] and when all edge weights are natural numbers.

### 2.7.1 Subset-Weight Optimization Problems

Next, we introduce a class of optimization problems we consider in Chapters 3 and 7.

**Definition 2.11.** An NP-optimization problem $L$ is a *subset-weight optimization problem* if $L$ can be expressed by

- a polynomial-time computable function $U$ that maps each instance $I$ of $L$ to a universe $U(I)$, such that each feasible solution of $I$ is a subset of $U(I)$,
- a polynomial-time computable function $f$ which checks for an instance $I$ of $L$ and a set $S \subseteq U(I)$ if $S$ is a feasible solution for $I$, and
- a polynomial-time computable weight function $\omega$ which assigns a nonnegative rational weight to each pair $(I, u)$, where $I$ is an instance of $L$ and $u$ is an element of $U(I)$.

Furthermore, if $L$ is a minimization (maximization) problem, one wants to find a feasible solution $S$ for $I$ of minimum (maximum) weight, where the weight of $S$ is defined as $\omega(I, S) := \sum_{u \in S} \omega(I, u)$.

In this work, we only consider subset-weight optimization problems, where each element receives a natural number as its weight.

Weighted Independent Set is a subset-weight maximization problem: the feasible solutions are the independent sets of the input graph $G$, these are all subsets of the vertex set $V$ (the universe), one can check in polynomial time if a vertex set $S$ is an independent set, and the total weight of $S$ is defined as the sum of the

weights of the vertices of $S$. Similarly, Weighted Vertex Cover—the problem where we want to find a vertex cover of minimum total weight—is a subset-weight minimization problem.

While this may not be obvious at first, Max Cut can also be viewed as a subset-weight minimization problem: We may take the universe as the edge set and the feasible solutions are exactly those edge sets $S \subseteq E$ such that $G$ has a partition $(A, B)$ with $E_G(A, B) = S$ (which can be decided in polynomial time for a given set $S$).

We also want to remark that there is a close relation between subset-weight minimization problems and subset-weight maximization problems. Namely, for each subset-weight minimization problem there is a dual subset-weight maximization problem and vice versa. This is formalized by the following definition.

**Definition 2.12.** Let $L$ be a subset-weight optimization problem consisting of functions $U$, $f$, and $\omega$. The *dual subset-weight optimization problem of* $L$ is the subset-weight optimization problem $L'$ consisting of functions $U'$, $f'$, $g'$, and $\omega'$, where $\mathcal{D}_L = \mathcal{D}_{L'}$, that is, $L$ and $L'$ have the same instances, and where for each instance $I$ of $L$

- $U(I) = U'(I)$,
- for each element $u \in U(I)$, $\omega(I, u) = \omega'(I, u)$,
- for each set $S \subseteq U(I)$, $f(I, S) = f'(I, U(I) \setminus S)$, that is, $S$ is a solution for $I$ with respect to $L$ if and only if the complement set $U(I) \setminus S$ is a solution for $I$ with respect to $L'$,

and if $L$ is a subset-weight minimization problem and $L'$ is a subset-weight maximization problem, or vice versa.

By definition, if $L'$ is the dual subset-weight optimization problem of $L$, then $L$ is the dual subset-weight optimization problem of $L'$. Hence, we may simply say that $L$ and $L'$ are *dual*. For example, Weighted Independent Set and Weighted Vertex Cover are dual subset-weight optimization problems. Note that two dual subset-weight optimization problems $L$ and $L'$ are computationally equivalent, that is, for each instance $I$ of these problems, one can find an optimal solution for $I$ with respect to $L$ in polynomial time if and only if one can find an optimal solution for $I$ with respect to $L'$ in polynomial time.

### 2.7.2 Swap Neighborhoods

Next, we define the arguably most natural scalable neighborhood for subset-weight optimization problems.

**Definition 2.13.** Let $L$ be a subset-weight optimization problem, let $I$ be an instance of $L$, and let $S \subseteq U(I)$ be a feasible solution for $I$. A *$k$-swap*, for $k \in \mathbb{N}$, is a subset $W \subseteq U(I)$ of size at most $k$.

We say that $W$ is *valid for $S$ in $I$* if $S \oplus W$ is also a feasible solution for $I$. We say that two feasible solutions $S$ and $S'$ for $I$ are *$k$-(swap-)neighbors in $I$* if $W := S \oplus S'$ is a $k$-swap. Additionally, we say that $S'$ is an *improving $k$-neighbor of $S$ in $I$* and that $W$ is an *improving $k$-swap* if the total weight of $S'$ is better than the total weight of $S$. If there is no improving $k$-neighbor of $S$ in $I$, then $S$ is *$k$-optimal for $I$*.

**Definition 2.14.** Let $S$ be a subset of $U(I)$ and let $k^{\text{in}}$ and $k^{\text{out}}$ be nonnegative natural numbers. A set $W \subseteq U(I)$ is a *$(k^{\text{in}}, k^{\text{out}})$-swap for $S$*, if $|W \setminus S| \leq k^{\text{in}}$ and if $|W \cap S| \leq k^{\text{out}}$.

Informally, $W$ adds at most $k^{\text{in}}$ vertices to $S$ and removes at most $k^{\text{out}}$ vertices from $S$. Note that for $k \geq k^{\text{in}} + k^{\text{out}}$, a $(k^{\text{in}}, k^{\text{out}})$-swap for $S$ is a $k$-swap. Similar to $k$-swaps, we also define the notions of *valid $(k^{\text{in}}, k^{\text{out}})$-swaps*, *improving $(k^{\text{in}}, k^{\text{out}})$-swaps*, *$(k^{\text{in}}, k^{\text{out}})$-neighbors*, *improving $(k^{\text{in}}, k^{\text{out}})$-neighbors*, and *$(k^{\text{in}}, k^{\text{out}})$-optimality*.

For constant values of $k$, every subset-weight optimization problem with the $k$-swap neighborhood is contained in PLS, since for a given feasible solution one can enumerate all feasible solutions within the $k$-swap neighborhood in polynomial time. Moreover, the relation between dual subset-weight optimization problems $L$ and $L'$ directly implies that one can easily derive PLS-reductions between $L$ and $L'$ with respect to swap neighborhoods.

**Observation 2.15.** *Let $L$ and $L'$ be dual subset-weight optimization problems. For each $k \in \mathbb{N}$, there is a linear-time computable* PLS*-reduction from $L/k$-swap to $L'/k$-swap. For each $k^{\text{in}} \in \mathbb{N}$ and each $k^{\text{out}} \in \mathbb{N}$, there is a linear-time computable* PLS*-reduction from $L/(k^{\text{in}}, k^{\text{out}})$-swap to $L'/(k^{\text{out}}, k^{\text{in}})$-swap.*

Let $L$ and $L'$ be dual subset-weight optimization problems. The correctness of Observation 2.15 with respect to $(k^{\text{in}}, k^{\text{out}})$-swaps relies on the fact that for each instance $I$ of $L$ and each vertex set $S \subseteq U(I)$, $S$ is a solution for $I$ with respect to $L$ if and only if $U(I) \setminus S$ is a solution for $I$ with respect to $L'$. Hence, a valid improving $(k^{\text{in}}, k^{\text{out}})$-swap for $S$ in $I$ with respect to $L$ is a valid improving $(k^{\text{out}}, k^{\text{in}})$-swap for $U(I) \setminus S$ in $I$ with respect to $L'$. Note that Observation 2.15 implies that $L/k$-swap is PLS-hard if and only if $L'/k$-swap is PLS-hard. The same also holds for PLS-completeness. Furthermore, we can find a $(k^{\text{in}}, k^{\text{out}})$-optimal solution for $L$ in polynomial time if and only if we can find a $(k^{\text{out}}, k^{\text{in}})$-optimal solution for $L'$ in polynomial time. Hence, to show hardness results or polynomial-time algorithms to find locally optimal solutions for $L$ or $L'$ with respect to swap neighborhoods, we only need to consider one of $L$ or $L'$.

## 2.8 An Algorithm for Searching the $k$-Flip Neighborhood

We now describe a black-box algorithm to find—for several optimization problems—improving solutions with respect to the $k$-flip neighborhood. More precisely, we consider NP-optimization problems, where for each instance $I$ of $L$, there is a polynomial-time computable universe $U(I)$ such that each solution of $I$ is (or can be expressed as) a coloring of $U(I)$. Among others, this includes problems like Max Cut and Max Sat, but also subset-weight optimization problems. For a subset-weight optimization problem, a solution $S \subseteq U(I)$ can be expressed as the 2-coloring that assigns color 1 to each element of $S$ and color 2 to all other elements of $U(I)$.

We now provide the definitions that are necessary to present our algorithm. Let $\chi\colon U(I) \to \mathbb{N}$ be a solution for $I$ and let $k \in \mathbb{N}$. We say that a collection $\mathcal{U}$ of subsets of $U(I)$ (of size at most $k$ each) is a *sufficient candidate collection for* $\chi$, if there is a better solution $\chi'$ for $I$ with $d_{\text{flip}}(\chi, \chi') \leq k$ if and only if there is a *candidate* $S \in \mathcal{U}$ such that there is a better coloring $\chi^*$ for $I$ with $D_{\text{flip}}(\chi, \chi^*) = S$. Note that the collection of all subsets of $U(I)$ of size at most $k$ is a trivial sufficient candidate collection for each solution for $I$. Moreover, note that determining whether there is a better solution in the $k$-flip neighborhood of $\chi$ can be done by performing two steps: Compute a sufficient candidate collection $\mathcal{U}$ for $\chi$ and check for each candidate $S \in \mathcal{U}$, whether there is a better solution $\chi^*$ for $I$ with $D_{\text{flip}}(\chi, \chi^*) = S$. In the following, we describe a general approach to compute a sufficient candidate collection $\mathcal{U}$ for $\chi$ based on some auxiliary support graph.

Let $G_{\text{sup}}$ be a graph with vertex set $U(I)$. We say that $G_{\text{sup}}$ is a *candidate support graph for* $\chi$ if the collection of all connected vertex sets of size at most $k$ in $G_{\text{sup}}$ is a sufficient candidate collection for $\chi$. Here, the constraint that the considered vertex sets are connected comes from the intuitive idea that if the flip of a vertex set $S$ might yield a better solution, then the vertices of $S$ should interact with each other in the candidate support graph. Note that the complete graph on the vertex set $U(I)$ is always a candidate support graph for $\chi$. Since the connected vertex sets of size at most $k$ of a graph $G$ with $n$ vertices can be enumerated in $(e \cdot \Delta(G))^k \cdot k \cdot n$ time [108], we thus obtain an efficient algorithm to compute a sufficient candidate collection for $\chi$, if a given candidate support graph has a small maximum degree.

**Lemma 2.16.** *Let $I$ be an instance of an* NP*-optimization problem where each solution of $I$ is a coloring of $U(I)$. Moreover, let $\chi$ be a solution for $I$, and let $G_{\text{sup}}$ be a given candidate support graph for $\chi$. Then, one can compute a sufficient candidate collection for $\chi$ of size $(e \cdot \Delta(G_{\text{sup}}))^k \cdot k \cdot |U(I)|$ in the same asymptotic running time.*

Since we aim to find a sufficient candidate collection as small as possible, we ideally want to consider candidate support graphs of rather small maximum degree.

In combination with an algorithm $A$ that determines for a given candidate $S \subseteq U(I)$ of size at most $k$, whether there is a better solution $\chi^*$ for $I$ with $D_{\text{flip}}(\chi, \chi^*) = S$, we obtain the following black-box algorithm.

**Theorem 2.17.** *Let $I$ be an instance of an* NP*-optimization problem where each solution of $I$ is a coloring of $U(I)$. Let $\chi$ be a solution for $I$ and let $G_{\text{sup}}$ be a given candidate support graph for $\chi$. One can determine in time $(e{\cdot}\Delta(G_{\text{sup}}))^k{\cdot}f(A){\cdot}|U(I)|+|I|^{\mathcal{O}(1)}$, whether there is a better solution $\chi'$ for $I$ with $d_{\text{flip}}(\chi, \chi') \leq k$. Here, $f(A)$ is the running time of an algorithm $A$ that determines for any candidate $S \subseteq U(I)$ of size at most $k$, whether there is a better solution $\chi^*$ for $I$ with $D_{\text{flip}}(\chi, \chi^*) = S$.*

We make use of this algorithm in Chapters 4 and 5. Note that if each solution for $I$ can be expressed as a $c$-coloring for some $c \geq 2$, then we can provide an algorithm $A$ that determines for a given candidate $S \subseteq U(I)$ of size at most $k$, whether there is a better solution $\chi^*$ for $I$ with $D_{\text{flip}}(\chi, \chi^*) = S$: This can be done for example by a branching algorithm that considers for each element $s \in S$ all possible colors of $[1, c] \setminus \{\chi(s)\}$ that the element $s$ may receive under $\chi^*$. Since such a branching algorithm runs in $(c-1)^k \cdot |I|^{\mathcal{O}(1)}$ time, this then yields the following result.

**Theorem 2.18.** *Let $c \geq 2$. Let $I$ be an instance of an* NP*-optimization problem where each solution of $I$ is a $c$-coloring of $U(I)$. Moreover, let $\chi$ be a solution for $I$ and let $G_{\text{sup}}$ be a given candidate support graph for $\chi$. Then, one can determine in $(e \cdot \Delta(G_{\text{sup}}))^k \cdot k \cdot (c-1)^k \cdot |U(I)| + |I|^{\mathcal{O}(1)}$ time, whether there is a better solution $\chi'$ for $I$ with $d_{\text{flip}}(\chi, \chi') \leq k$.*

In addition to its applications in Chapters 4 and 5, Theorem 2.18 has implications for example for $d_{\text{flip}}$-LS Max Sat which was first analyzed by Szeider [161] under the name of $k$-Flip Max Sat. Szeider [161] showed that, for a Boolean formula $F$, there is a polynomial-time computable graph $G$ with vertex set equal to the variable set of $F$ such that $G$ fulfills the properties of a candidate support graph for $d_{\text{flip}}$-LS Max Sat. Szeider further showed that this graph $G$ has maximum degree at most $pq$, where $p$ denotes the maximum number of occurrences of any variable in $F$ and $q$ denotes the size of the largest clause in $F$. Based on this observation, Szeider concluded that $d_{\text{flip}}$-LS Max Sat can be solved in $2^{p \cdot q \cdot k} \cdot |F|^{\mathcal{O}(1)}$ time. The presented candidate support graph by Szeider [161] together with Theorem 2.18 imply the following even better running time for $d_{\text{flip}}$-LS Max Sat, since each truth assignment can be interpreted as a 2-coloring.

**Corollary 2.19.** *$d_{\text{flip}}$-LS Max Sat can be solved in $(e \cdot p \cdot q)^k \cdot n^{\mathcal{O}(1)}$ time.*

# Chapter 3

# Parameterized Local Search for Vertex Cover

Vertex Cover is the most prominent and well-studied problem in parameterized complexity [35, 44, 84] and from the perspective of parameterized local search [52, 69, 98]. Recall that in Vertex Cover, the set of feasible solutions of a graph $G = (V, E)$ is the collection of vertex covers of $G$, that is, vertex sets $S \subseteq V$ that cover all edges of the graph. When applying local search for Vertex Cover, the most obvious choice for a local search neighborhood is the $k$-swap neighborhood.

The problem of deciding whether a given vertex cover $S$ of a graph $G$ has a smaller vertex cover in its $k$-swap neighborhood, called LS Vertex Cover, is W[1]-hard with respect to $k$ [52]. There are, however, some positive results for LS Vertex Cover. In particular, LS Vertex Cover admits an FPT-algorithm for $\Delta(G) + k$, where $\Delta(G)$ is the maximum degree of the input graph [52]. That is, it can be solved in $f(\Delta(G), k) \cdot n^{\mathcal{O}(1)}$ time. While this running time bound is certainly interesting for bounded-degree graphs, it does not necessarily deliver on the promise of parameterized local search that the superpolynomial part of the running time depends mostly on $k$: for example $f(\Delta(G), k)$ could be $2^{\Delta(G) \cdot k}$. However, it is known that LS Vertex Cover can be solved in time $\mathcal{O}(2^k \cdot (\Delta(G)-1)^{k/2} \cdot k^3 \cdot n)$ [98]. In this running time only $k$ appears in the exponent and $\Delta(G)$ appears only in the base of the exponential function. Consequently, for small values of $k$ the running time guarantee can still be practically relevant, even when $\Delta$ is not too small. In particular, the running time is polynomial for every fixed $k$. Note that the running time of this algorithm grows strongly with respect to $k$ and only mildly with respect to $\Delta(G)$ (see Definition 2.8). The usefulness of this algorithm was confirmed by experiments which showed that the problem can be solved efficiently for $k$ up to 25 [98].

**Our results.** We continue the research on parameterized local search for Vertex Cover. We provide FPT-algorithms for LS Vertex Cover parameterized by $k$ and several structural parameters $\ell$ of $G$. We consider the treewidth of the input graph $G$, denoted by $\mathrm{tw}(G)$, the $h$-index of the input graph $G$, denoted by $h(G)$, and the modular-width of $G$, denoted by $\mathrm{mw}(G)$. In all our FPT-algorithms, the running time grows strongly with respect to $k$ and only mildly with respect to the particular structural parameter. Moreover, for all these algorithms, the running time depends only linearly on the size of the input graph.

The most general of our algorithms actually solve Gap LS Weighted Vertex Cover which differs from LS Vertex Cover in two ways: Firstly, the vertices of the graph are weighted and the goal is to find a vertex cover of smaller weight in the $k$-swap neighborhood. Secondly, the weight of the new vertex cover should be better by at least a given threshold $d$. Since local search approaches for Weighted Vertex Cover have been studied from a more practical perspective, analyzing such a weighted variant of LS Vertex Cover is well motivated. In addition, using a gap-variant of local search could reduce the number of necessary local improvements before one finds a local optimum. This is important for weighted local search problems, since the number of such improvement steps may be exponential [95] even for swaps of constant size (see Chapter 7) when only asking for any better solution. We now discuss the results in detail.

The $h$-index of a graph $G$ is the largest number $h$ such that $G$ has at least $h$ vertices with degree at least $h$ [48]. For Gap LS Weighted Vertex Cover, we obtain an algorithm with running time $\mathcal{O}(k! \cdot (h(G)-1)^k \cdot n)$. This can be seen as an improvement over the FPT-algorithm for $\Delta(G)$ and $k$ [98], since $h(G)$ is never larger than $\Delta(G)$. In fact, in many real-world instances the input graphs are scale-free, and on scale-free graphs $h(G)$ is drastically smaller than $\Delta(G)$. Even in such graphs, in order to speak of an improvement, it is imperative that the running time of the FPT-algorithm grows mildly with respect to $h(G)$ and strongly with respect to $k$: a running time of $\mathcal{O}(2^{h(G)\cdot k} \cdot n)$ would be less desirable than the previous one for $\Delta(G)$ and $k$ since the exponent would not be confined to the search radius $k$.

For Gap LS Weighted Vertex Cover the FPT-algorithm for $\mathrm{tw}(G)$ and $k$ has running time $\mathcal{O}((\mathrm{tw}(G)^{3k} + k^2) \cdot n)$. It is based on dynamic programming on the tree decomposition. The main observation is that for a bag of the tree decomposition it is sufficient to consider all possibilities of how an improving swap interacts with the bag. For LS Vertex Cover we further reduce the running time to $\mathcal{O}((\mathrm{tw}(G)^{3\cdot\lceil\frac{k}{2}\rceil} + k^2) \cdot n)$ by observing that we only need to consider small interactions with the bags of the tree decomposition. Hence, compared to the algorithm for Gap LS Weighted Vertex Cover, we are able to consider swaps of double the size.

We then consider the modular-width. This parameter measures a different structural aspect, the similarity of neighborhoods in the graph, than treewidth or the degree-related parameterizations. In particular, the modular-width can be very small in dense graphs. For Gap LS Weighted Vertex Cover we develop an FPT-algorithm with running time $\mathcal{O}(\mathrm{mw}(G)^k \cdot k \cdot (\mathrm{mw}(G)+k) \cdot n + m)$, where $\mathrm{mw}(G)$ is the modular-width of $G$. The algorithm is based on bottom-up dynamic programming over the decomposition and considers all possibilities how an improving swap may interact with a node of the modular decomposition.

We complement these algorithms by conditional lower bounds that are based on the assumption that matrix-multiplication-based algorithms for Clique are running-time optimal [1]. We show that under this assumption, we may not expect a very large improvement over the previously known and new algorithms.

**Problem-specific notation.** In the following, we formally define the parameterized local search problems we consider in this chapter.

**Definition 3.1.** Let $G$ be a graph, let $S$ be a vertex cover of $G$, let $\omega\colon V(G) \to \mathbb{N}$ be a weight function, and let $W$ be a swap. The *improvement* of $W$ is defined as $\delta_\omega^S(W) := \omega(W \cap S) - \omega(W \setminus S)$. Moreover, $W$ is *improving* if $\delta_\omega^S(W) > 0$ and *$d$-improving* for some $d \in \mathbb{N}$ if $\delta_\omega^S(W) \geq d$.

If $S$ or $\omega$ are clear from the context, we may omit them. Next, we formally define the local search problems for Vertex Cover that we consider in this work.

LS Weighted Vertex Cover (LS WVC)
**Input:** A graph $G = (V, E)$, a weight function $\omega\colon V \to \mathbb{N}$, a vertex cover $S$ of $G$, and $k \in \mathbb{N}$.
**Question:** Is there a valid improving $k$-swap $W \subseteq V$ for $S$ in $G$?

Gap LS Weighted Vertex Cover (GLS WVC)
**Input:** A graph $G = (V, E)$, a weight function $\omega\colon V \to \mathbb{N}$, a vertex cover $S$ of $G$, $k \in \mathbb{N}$, and $d \in \mathbb{N}$.
**Question:** Is there a valid $d$-improving $k$-swap $W \subseteq V$ for $S$ in $G$?

Moreover, we define Gap LS Vertex Cover (GLS VC) as the special case of GLS WVC where $\omega(v) = 1$ for each $v \in V$ and $d \in [1, k]$, and LS Vertex Cover (LS VC) as the special case of GLS VC, where $d = 1$. Note that for an instance of GLS VC, the improvement of a swap $W$ is $|W \cap S| - |W \setminus S|$. Let $I = (G, S, \omega, k, d)$ be an instance of GLS WVC. We say that $W \subseteq V(G)$ *is a*

*good swap for $I$*, if $W$ is a valid $d$-improving $k$-swap for $S$ in $G$. Further, we say that a good swap $W$ for $I$ is *minimal*, if each proper subset $W'$ of $W$ is not a good swap for $I$ and we say that $W$ is a *minimum* good swap for $I$ if there is no good swap $W'$ for $I$ with $|W'| < |W|$.

**Observation 3.2.** *Let $I := (G, S, \omega, k, d)$ be an instance of* GLS WVC*, let $W$ be a good swap for $I$, and let $X \subseteq W \cap S$. Then $W \setminus X$ is a valid $k$-swap for $S$ in $G$ with $\delta(W \setminus X) = \delta(W) - \delta(S)$.*

## 3.1 Basic Observations and Lower Bounds

In this section, we provide some basic observations about the considered local search problems and prove two conditional running time lower bounds.

Next, we first define the notion of swap-instances. These are instances obtained from an instance $I$ of GLS WVC after applying some partial swap. Swap-instances will be useful for describing certain parts of our algorithms such as branching rules. We then make some observations on certain useful properties of improving swaps. Those are mostly generalizations of known results for LS VC. Finally, we present our running time lower bounds for the considered parameters.

**Swap-instances.** In our algorithms, we may change instances by performing some partial swaps, for example during branching. We call the instance obtained by such an operation a *swap-instance*. Intuitively, the swap-instance $\text{swap}(I, W)$ for an instance $I$ of GLS WVC and a (partial) swap $W$ is the GLS WVC-instance obtained as follows: First, swap $W$. Note that $W$ might not be a valid swap. Thus, we swap further vertices to again obtain a vertex cover. To simplify the instance, the set $W' \supseteq W$ of swapped vertices is then removed from the instance together with the neighbors of $W \cap S$ in $S$. Finally, to maintain equivalence, the remaining budget $k$ is decreased by the number of swapped vertices and the required improvement $d$ is decreased by the improvement of $W'$.

The unique inclusion-minimal superset $W'$ of $W$ that has to be swapped to again obtain a vertex cover is called the *extension of $W$ with respect to $I$*. Note that the extension of $W$ with respect to $I$ is exactly $W \cup (N(W) \setminus S)$, since each independent set vertex adjacent with at least one vertex of $S$ that is swapped out of the vertex cover, has to be swapped to obtain a vertex cover.

Formally, a swap-instance is defined as follows.

**Definition 3.3.** Let $I = (G, \omega, S, k, d)$ be an instance of GLS WVC and let $W \subseteq V(G)$ be a $k$-swap. Let $W' := W \cup (N(W) \setminus S)$ be the extension of $W$ with respect

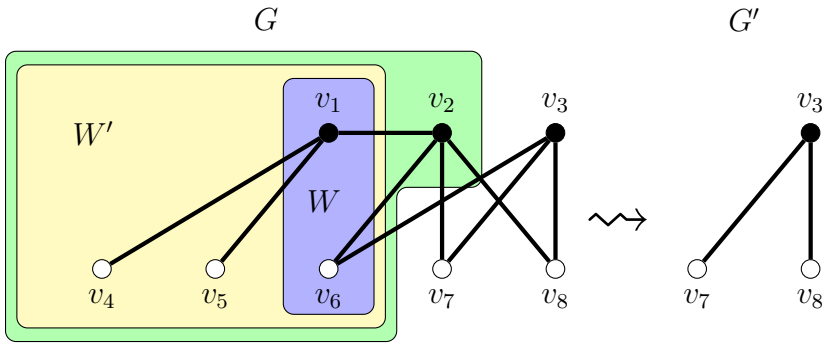



**Figure 3.1:** An instance $I := (G, S, k, d)$ of GLS VC (left) and the swap-instance $\text{swap}(I, W) := (G', S', k', d')$ (right) obtained from the swap $W := \{v_1, v_6\}$. The vertex cover vertices are black, the independent set vertices are white. The green area contains the vertices of $N(W \cap S) \cup W'$ which are in $G$ but not in $G'$. Since $W'$ has size 4 and contains only one vertex of $S$, $k' := k - 4$ and $d' := d + 2$. Moreover, the vertex $v_2$ is not contained in $G'$ since $v_1$ is adjacent to $v_2$ and leaves the vertex cover, which implies that $v_2$ cannot leave the vertex cover afterwards.

to $I$. The instance

$$\text{swap}(I, W) := (G', \omega', S' := S \setminus W, k', d')$$

with $G' := G - (N(W \cap S) \cup W')$, $k' := k - |W'|$, $d' := d - \delta(W') = d - \omega(W' \cap S) + \omega(W' \setminus S)$, and where $\omega'$ is the restriction of $\omega$ to $V(G')$ is the *swap-instance for $I$ and $W$*.

An example of a swap-instance can be seen in Figure 3.1.

**Lemma 3.4.** *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of* GLS WVC *and let $W \subseteq V$ be a vertex set such that $W \cap S$ is an independent set. There is a good swap $W^*$ for $I$ with $W \subseteq W^*$ if and only if* $\text{swap}(I, W)$ *is a yes-instance of* GLS WVC.

*Proof.* Let $(G', \omega', S', k', d') := \text{swap}(I, W)$ and let $W'$ be the extension of $I$ with respect to $I$, that is, $W' = W \cup (N(W) \setminus S)$.

Let $W^*$ be a good swap for $I$ with $W \subseteq W^*$. Since $W^*$ is valid, $W' \subseteq W^*$. Moreover, no vertex of $W^* \cap S$ has a neighbor in $W \cap S = W' \cap S$. Hence, $W^* \setminus W'$ contains only vertices of $G'$, since $G'$ contains all vertices of $G$ except for the vertices of $N(W \cap S) \cup W'$. Moreover, $W^* \setminus W'$ has size $|W^*| - |W'| \leq k - |W'| = k'$,

and $\delta(W^* \setminus W') = \delta(W^*) - \delta(W') \geq d - \delta(W') = d'$. As a consequence, $W^* \setminus W'$ is a good swap for $\text{swap}(I, W)$.

Let $X$ be a good swap for $\text{swap}(I, W)$ and let $W^* := X \cup W'$. By definition, $W^*$ has size at most $k$, $W^*$ is $d$-improving, and $W$ is a subset of $W^*$. Thus, it remains to show that $W^*$ is a valid swap for $S$ in $G$. Since (i) $W'$ is a valid swap for $S$ in $G$, (ii) $X$ is a valid swap for $S'$ in $G'$, and (iii) $G'$ contains all vertices of $(V \setminus W') \setminus S$, we have the following: for each vertex $v \in W^* \cap S$, each neighbor of $v$ in $V \setminus S$ is contained in $W^*$. Hence, $W^*$ is valid if there are no two adjacent vertices of $S$ in $W^*$. Since $W' \cap S$ and $X \cap S$ are both independent sets in $G$ and since no vertex of $V(G') \cap S \supseteq X \cap S$ is adjacent to some vertex of $W' \cap S$, this property holds. Consequently, $W^*$ is valid. □

Consider the swap-instance of an instance $I := (G, S, k, d)$ of the unweighted problem GLS VC and some $W \subseteq V(G)$. Then $k' + d' = k + d - 2 \cdot |W \cap S|$, since $\delta(W') = -|W'| + 2 \cdot |W \cap S|$. This observation has the following influence for our branching algorithms: If in each branching step, we swap at least one vertex out of the vertex cover, the depth of the branching-tree is at most $\frac{k+d}{2}$. Let $I$ be an instance of GLS WVC and let $W$ be the subset of some valid swap. When we replace the instance $I$ by $\text{swap}(I, W)$ we may say that we *swap $W$ in $I$*.

**Properties of improving swaps.** Next, we generalize some known properties of good swaps of LS VC to the more general problems GLS VC and LS WVC. Consider some improving swap $W$ for $S$ in $G$. Then, each connected component in $G[W]$ is a valid swap and since $W$ is improving, at least one connected component in $G[W]$ is an improving swap for $S$ in $G$. Hence, the following holds.

**Observation 3.5.** *Let $I = (G, \omega, S, k)$ be an instance of* LS WVC*. If $I$ is a yes-instance of* LS WVC*, then there is a good swap $W$ for $I$ such that $W$ is connected.*

Next, we show that for the unweighted problem GLS VC it is sufficient to consider instances where $k + d$ is even.

**Lemma 3.6.** *Let $I = (G, S, k, d)$ be an instance of* GLS VC *where $k + d$ is odd and $k \geq 1$. Then, $I$ is a yes-instance of* GLS VC *if and only if $I' := (G, S, k-1, d)$ is a yes-instance of* GLS VC*.*

*Proof.* ($\Rightarrow$) Let $W$ be a minimal good swap for $I$. Since $d \geq 1$, $W$ contains a vertex $v$ of $S$. Consider the swap $W' := W \setminus \{v\}$. Since $W$ is a $k$-swap and $S \oplus W' = (S \oplus W) \cup \{v\}$, $W'$ is a valid $k$-swap for $S$ in $G$. By the fact that $W$ is a minimal good swap for $I$, $W'$ is not a good swap for $I$. Hence, $W'$ is not $d$-improving, since

we already showed that $W$ is a valid $k$-swap for $S$ in $G$. This implies that $\delta(W) = d$, since $\delta(W) \geq d > \delta(W') = \delta(W) - 1$. Next, we show that $W$ has size at most $k-1$ which then implies that $W$ is a good swap for $I'$.

Recall that the improvement of $W$ is $\delta(W) = |W \cap S| - |W \setminus S| = |W| - 2 \cdot |W \setminus S|$. Hence, $|W|$ is odd if and only if $\delta(W)$ is odd, that is, $|W| + \delta(W)$ is even. Recall that $\delta(W) = d$ and that $k + d$ is odd. Consequently, $|W| + \delta(W) = |W| + d \leq k + d$ implies that $W$ has size less than $k$. Hence, $W$ is a good swap for $I'$.

($\Leftarrow$) Each $(k-1)$-swap is a $k$-swap. Hence, each good swap for $I'$ is a good swap for $I$. □

Some of our algorithms branch over all possible intersections of a $d$-improving $k$-swap $W$ with a given vertex set $X$. The following lemma shows that for GLS VC, we only have to consider intersections of size at most $\frac{k+d}{2}$ of $X$ with a potential improving swap $W$. Namely, we only have to consider vertices $S_X$ of $W \cap X$ in $S$ and the vertices $C_X$ of $W \cap X$ in $V \setminus S$ that are not contained in the extension of $S_x$.

**Lemma 3.7.** *Let $I = (G, S, k, d)$ be an instance of* GLS VC *and let $W$ be a good swap for $I$. Further, for a set of vertices $X \subseteq V(G)$, let $S_X := W \cap X \cap S$ and $C_X := W \cap X \setminus N[S_X]$. If $|S_X \cup C_X| > \frac{k+d}{2}$, then there is a good swap $W'$ for $I$ such that $W'$ is a proper subset of $W$.*

*Proof.* First, we show that $W^* := S_X \cup (N(S_X) \setminus S)$ is a good swap for $I$. Note that $W^*$ contains no vertex of $C_X$ and $W^*$ is a (not necessarily proper) subset of $W$, since $W$ is a valid swap for $S$ in $G$ and contains all vertices of $S_X$. By definition, $W^*$ is a valid $k$-swap for $S$ in $G$. It remains to show that $W^*$ is $d$-improving. To this end, note that each $d$-improving $k$-swap contains at most $\frac{k-d}{2}$ vertices of $V \setminus S$. In particular, $|W \setminus S| = |C_X| + |N(S_X) \setminus S| \leq \frac{k-d}{2}$. Since $|S_X \cup C_X| > \frac{k+d}{2}$, this implies $|S_X| > d + \frac{k-d}{2} - |C_X| \geq d + |(N(S_X) \setminus S)|$. Hence, $W^*$ is $d$-improving and thus a good swap for $I$.

If $W^*$ is a proper subset of $W$, then the statement already holds. Hence, assume in the following that $W = W^*$. This then implies that $C_X = \emptyset$. As a consequence, $S_X$ has size more than $\frac{k+d}{2}$. Note that this implies that the improvement of $W$ is at least $d + 1$, since $|W \setminus S_X| = |W| - |S_X| < k - \frac{k+d}{2} = \frac{k-d}{2}$. Let $v$ be an arbitrary vertex of $S_X$. Consider the swap $W' := W \setminus \{v\}$. Due to Observation 3.2, $W'$ is a valid $k$-swap for $S$ in $G$ with $\delta(W') = \delta(W) - 1 \geq d$. Hence, $W'$ is a good swap for $I$. Moreover, since $W'$ is a proper subset of $W$, the statement holds. □

To obtain FPT running times that are linear in the input size, if the considered parameters are constant, we handle instances with small values of $k$ separately.

**Lemma 3.8.** GLS WVC *can be solved in* $\mathcal{O}(n+m)$ *time if* $k \leq 2$ *and* GLS VC *can be solved in* $\mathcal{O}(n+m)$ *time if* $k+d \leq 4$.

To show Lemma 3.8, we show the two statements separately. First, we show the statement for the weighted problem.

**Lemma 3.9.** GLS WVC *can be solved in* $\mathcal{O}(n+m)$ *time if* $k \leq 2$. *Moreover, for* $k \leq 2$, *a valid* $k$*-swap of maximum improvement can be found in* $\mathcal{O}(n+m)$ *time.*

*Proof.* Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS WVC with $k \leq 2$. We first compute the set $S^* := \{v \in S \mid N(v) \subseteq S\}$ of vertex cover vertices without independent set neighbors in $\mathcal{O}(n+m)$ time. Moreover, if $S^* \neq \emptyset$, then we also compute the vertex $v^* \in S^*$ that maximizes $\omega(v^*)$ in $\mathcal{O}(n)$ time.

If $k = 1$, then each good swap for $I$ consists of a single vertex in $S^*$. Hence, $I$ is a no-instance of GLS WVC if $S^* = \emptyset$. Otherwise, $\{v^*\}$ is a valid 1-swap of maximal improvement for $I$. Consequently, $I$ is a yes-instance of GLS WVC if and only if $\omega(v^*) \geq d$ and, thus, GLS WVC can be solved in $\mathcal{O}(n+m)$ time if $k = 1$.

If $k = 2$, then we additionally compute the set $\mathcal{W}^* := \{\{v, w\} \mid v \in S, N(v) \setminus S = \{w\}, \omega(v) > \omega(w)\}$ in $\mathcal{O}(n+m)$ time. Moreover, if $\mathcal{W}^* \neq \emptyset$, then we also compute the swap $W^* \in \mathcal{W}^*$ that maximizes $\delta(W^*)$ in $\mathcal{O}(n)$ time. Note that each minimal good swap either (i) consists of a single vertex in $S^*$, (ii) consists of two non-adjacent vertices in $S^*$, or (iii) is a swap in $\mathcal{W}^*$. Hence, if $\mathcal{W}^* \neq \emptyset$ and $\delta(W^*) \geq d$, then $I$ is a yes-instance of GLS WVC. Suppose that $\mathcal{W}^* = \emptyset$ or $\delta(W^*) < d$. If $S^* = \emptyset$, then $I$ is a no-instance of GLS WVC. Thus, suppose that $S^* \neq \emptyset$. If $\omega(v^*) \geq d$, then $I$ is a yes-instance of GLS WVC. Otherwise, $I$ is a yes-instance of GLS WVC if and only if there are two non-adjacent vertices of $S^*$ of total weight at least $d$.

In the following, we show that we can determine in $\mathcal{O}(n+m)$ time whether such a pair of vertices exists. Moreover, if such a pair exists, then we can find one in the same running time. Let $S_{\geq} := \{v \in S^* \mid \omega(v) \geq d/2\}$ be the vertices of $S^*$ of weight at least $d/2$ and let $S_{<} := \{v \in S^* \mid \omega(v) < d/2\}$ be the vertices of $S^*$ of weight less than $d/2$. These sets can be computed in $\mathcal{O}(n)$ time. Note that each good swap for $I$ contains at least one vertex of $S_{\geq}$. If there are two non-adjacent vertices $v$ and $w$ in $S_{\geq}$, then these vertices can be found in $\mathcal{O}(n+m)$ time by checking for each vertex $v$ of $S_{\geq}$ if $v$ has at most $|S_{\geq}| - 2$ neighbors in $S_{\geq}$. In this case, since $\omega(v) + \omega(w) \geq d$, $I$ is a yes-instance of GLS WVC. Otherwise, $S_{\geq}$ is a clique and $\mathcal{O}(|S_{\geq}|^2) \subseteq \mathcal{O}(m)$. Hence, we can sort the vertices of $S_{\geq}$ according to their weight in $\mathcal{O}(m)$ time. Since $S_{\geq}$ is a clique, each good swap for $I$ consists of one vertex $v$ of $S_{\geq}$ and one vertex $w$ of $S_{<}$ such that $\{v, w\} \notin E$. To find such vertices, we check for each $w \in S_{<}$, whether $w$ is adjacent to each vertex of $S_{\geq}$ and,

if this is not the case, whether $\omega(w) + \omega(v) \geq d$, where $v$ is the vertex with the highest weight in $S_{\geq} \setminus N(w)$. If the latter is true for some $v \in S_{<}$, $I$ is a yes-instance of GLS WVC. Otherwise, $I$ is a no-instance of GLS WVC. Since $S_{\geq}$ is sorted, we only have to consider the first $|N(w)|+1$ vertices of highest weight in $S_{\geq}$ to find the vertex $v \in S_{\geq} \setminus N(w)$ of highest weight. Hence, this last step can be performed in $\mathcal{O}(\sum_{w \in S_{<}}(|N(w)|+1)) \subseteq \mathcal{O}(n+m)$ time and, thus, GLS WVC can be solved in $\mathcal{O}(n+m)$ time if $k = 2$. □

Next, we show the statement for GLS VC.

**Lemma 3.10.** GLS VC *can be solved in* $\mathcal{O}(n+m)$ *time if* $k + d \leq 4$.

*Proof.* Let $I = (G = (V, E), S, k, d)$ be an instance of GLS VC with $k + d \leq 4$. If $k \leq 2$, then $I$ can be solved in $\mathcal{O}(n+m)$ time due to Lemma 3.9. Moreover, if $d = 0$, $I$ is a trivial yes-instance of GLS VC. Hence, in the following, we can assume that $k > 2$ and $d > 0$. Since $k + d \leq 4$, this then implies that $k = 3$ and $d = 1$. Hence, $I$ is a yes-instance of GLS VC if and only if (i) there is a vertex $v$ of $S$ with $N(v) \subseteq S$ or (ii) there are two non-adjacent vertices $w_1$ and $w_2$ of $S$ that have the same neighborhood in $V \setminus S$ and this neighborhood consists of a single vertex. In $\mathcal{O}(n+m)$ time, we can check if there is some vertex $v$ of $S$ with $N(v) \subseteq S$. If this is the case, answer yes. Otherwise, we search for the vertices $w_1$ and $w_2$. To this end, we remove all vertices of $S$ from $G$ that have at least two neighbors in $V \setminus S$ and store for all remaining vertices of $S$ the corresponding unique neighbor in $V \setminus S$. Since the vertices $w_1$ and $w_2$ we are looking for, have the same neighbor in $V \setminus S$, we can also remove all edges between vertices of $S$ that are adjacent to different vertices of $V \setminus S$. Hence, each connected component $C$ in the resulting graph $G'$ contains exactly one vertex $v_C$ of $V \setminus S$. More precisely, $v_C$ is adjacent to all other vertices of $C$. If at least one of the connected components $C$ is not a clique, then there are two non-adjacent vertices $w_1$ and $w_2$ of $C$ such that $N_G(w_1) \setminus S = N_{G'}(w_1) \setminus S = \{v_C\} = N_{G'}(w_2) \setminus S = N_G(w_2) \setminus S$. Hence, $\{w_1, v_C, w_2\}$ is a good swap for $I$ and, thus, $I$ is a yes-instance of GLS VC. Otherwise, if $C$ is a clique in $G'$ for each connected component in $G'$, then $I$ is a no-instance of GLS VC. Note that all described steps can be performed by iterating over the edges a constant number of times. Hence, this algorithm runs in $\mathcal{O}(n+m)$ time. □

**Lower bounds.** Let $\omega < 2.373$ be the matrix multiplication constant [4]. Using a reduction to matrix multiplication, one can solve the Clique problem, which asks whether an $n$-vertex graph has a clique of size $k$, in $\mathcal{O}(n^{\omega \cdot k/3})$ time [129]. It is a long-standing question whether this running time can be improved to $\mathcal{O}(n^{(\omega/3-\varepsilon)k})$ [1,176].

Assuming that this is not the case, we obtain the following lower bounds for our considered problem.

**Theorem 3.11.** *For every $\varepsilon > 0$ and every $d \in [1, k]$,* GLS VC *cannot be solved in $\mathcal{O}(\ell^{(\omega/3-\varepsilon)\cdot\frac{k+d}{2}})$ time where $\ell = \max\{n - \frac{k-d}{2}, \Delta(G), \mathrm{vc}(G), |S|, \mathrm{mw}(G)\}$, unless* Clique *can be solved in $\mathcal{O}(n^{(\omega/3-\varepsilon)\cdot k})$ time. This holds even for the permissive version.*

*Proof.* Let $\varepsilon > 0$ be a constant. We assume in the following that $\varepsilon < \omega/3$, since the statement follows directly for $\varepsilon \geq \omega/3$. Moreover, let $\widehat{I} = (\widehat{G} = (V, \widehat{E}), k)$ be an instance of Clique with $k \geq \frac{2}{(\omega/3)-\varepsilon}$ and let $n$ denote the size of $V$, and let $d$ be an arbitrary value between 1 and $k$. We show that we can compute in $\mathcal{O}(n^2)$ time an equivalent instance $I' = (G' = (V', E'), S, k', d)$ of GLS VC such that $\ell := \max\{|V'| - \frac{k'-d}{2}, \Delta(G'), \mathrm{vc}(G'), |S|, \mathrm{mw}(G')\}$ is at most $n$. First, let $G$ be the complement graph of $\widehat{G}$, that is, $G := (V, E)$ with $E := \binom{V}{2} \setminus \widehat{E}$. Note that a set $X \subseteq V$ is a clique in $\widehat{G}$ if and only if $X$ is an independent set in $G$ and that one can compute $G$ in $\mathcal{O}(n^2)$ time. We can assume that the maximum degree of $G$ is at most $|V| - k$, since vertices degree at least $|V| - k + 1$ are contained in no independent set of size $k$. We obtain $G'$ by adding a set $V^*$ of $k - d$ new vertices to $G$ such that $N_{G'}(v) = V$ for each vertex $v$ of $V^*$. Finally, we set $k' := 2k - d$ and $S := V$, which completes the construction of $I'$. Note that this takes at most $\mathcal{O}(n^2)$ time, since $k \leq n$. Next, we show that $\widehat{I}$ is a yes-instance of Clique if and only if $I'$ is a yes-instance of GLS VC.

($\Rightarrow$) Let $C \subseteq V$ be an independent set of size $k$ in $G$, then $S' := (V \setminus C) \cup V^*$ is a vertex cover of $G'$ such that $|S \oplus S'| = k'$ and $|S'| \leq |S| + |V^*| - |C| = |S| - d$. Consequently, $I'$ is a yes-instance of GLS VC.

($\Leftarrow$) Let $W$ be a good swap for $I'$ and let $S' := S \oplus W$. Since $W$ is a good swap for $I$, $W$ has size at most $k' = 2k - d$ and $|S'| < |S| - d$. Consequently, $C := S \setminus S' = W \cap S$ is non-empty. We show that $C$ is an independent set of size $k$ in $G$. Since $S'$ is a vertex cover of $G'$ and every vertex of $V^*$ is adjacent to every vertex of $V$, it follows that $W$ contains all vertices of $V^*$. By the fact that (i) $W$ has size at most $2k-d$, (ii) $W$ contains all vertices of $V^*$, and (iii) $V^*$ has size $k-d$, $W \cap S$ has size at most $k$. Moreover, since $|S'| \leq |S| - d$, $C$ has size at least $k' - |V^*| = k$. As a consequence, $C$ has size exactly $k$. Moreover, since $S'$ is a vertex cover of $G'$ and $S'$ contains no vertex of $C$, $C$ is an independent set in $G$. Consequently, $C$ is an independent set of size $k$ in $G$ and, thus, $\widehat{I}$ is a yes-instance of Clique.

Next, we show that $\ell := \max\{|V'| - \frac{k'-d}{2}, \Delta(G'), \mathrm{vc}(G'), |S|, \mathrm{mw}(G')\}$ is at most $n$. By construction, $|V'| = n + k - d = n + \frac{k'-d}{2}$. Since the maximum degree of $G$ is at most $n - k$, the maximum degree of $G'$ is at most $n$. Moreover, since $S$ is a

vertex cover of size $n$ of $G'$, $\mathrm{vc}(G') \leq n$. Next, we show that the modular-width of $G'$ is at most $n$. Since each vertex of $V$ is adjacent with each vertex of $V^*$ in $G'$, the modular-width of $G'$ is the maximum of the modular-width of $G'[V]$ and the modular-width of $G'[V^*]$ [124]. By the fact that $V^*$ is an independent set in $G'$, this implies that the modular-width of $G'$ is exactly the modular-width of $G'[V] = G$. Since the modular-width of any graph is never larger than the number of vertices of that graph [124], the modular-width of $G$ is thus at most $|V| = n$. Hence, the modular-width of $G'$ is at most $n$.

Now, if we have an algorithm $A$ solving GLS VC in $\mathcal{O}(\ell^{(\omega/3-\varepsilon)\cdot\frac{k+d}{2}})$ time, then Clique can be solved in $\mathcal{O}(n^{(\omega/3-\varepsilon)k})$ time as well: Since $k \geq \frac{2}{(\omega/3)-\varepsilon}$, the running time $\mathcal{O}(n^{(\omega/3-\varepsilon)\cdot k})$ dominates the time used to construct the instance $I'$ of GLS VC. Now the running time bound for solving Clique using $A$ follows directly from $\ell \leq n$ and $\frac{k'+d}{2} = k$. □

For the cases LS VC and GLS WVC, we obtain the following.

**Corollary 3.12.** *For every $\varepsilon > 0$,* LS VC *cannot be solved in $\mathcal{O}(\ell^{(\omega/3-\varepsilon)\cdot\lceil\frac{k}{2}\rceil})$ time for $\ell := \max\{n-k+1, \Delta(G), \mathrm{vc}(G), |S|, \mathrm{mw}(G)\}$ and* GLS WVC *cannot be solved in $\mathcal{O}(n^{(\omega/3-\varepsilon)\cdot k})$ time, unless* Clique *can be solved in $\mathcal{O}(n^{(\omega/3-\varepsilon)\cdot k})$ time.*

*Proof.* The statement for LS VC follows directly from Theorem 3.11 since LS VC is the case of GLS VC where $d = 1$. Moreover, the statement for GLS WVC follows from Theorem 3.11 by considering instances of GLS VC where $k = d$ which are restricted instances of GLS WVC where each vertex receives weight one. □

Note that the latter restriction of Theorem 3.11 and Corollary 3.12 implies that these running time lower bounds also hold for the *permissive* version of each considered problem.

Based on the same reduction, we also derive the following by the fact that Clique cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails [35].

**Corollary 3.13.** LS VC *cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. This holds even for the permissive version of* LS VC.

Since LS VC is a special case of GLS VC, LS WVC, and GLS WVC, this running time lower bound also transfers to these more general problems too.

## 3.2 Parameterization by Treewidth

In this section, we present FPT-algorithms for $k$ and the treewidth of $G$. As can be expected, the algorithms makes use of tree decompositions, for which we recall the definition in the following.

### 3.2.1 Definition and Notion of Tree Decomposition

A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{T}, \beta)$ consisting of a rooted tree $\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*)$ with root $x^* \in \mathcal{V}$ and a function $\beta \colon \mathcal{V} \to 2^V$ such that

1. for each vertex $v$ of $V$, there is at least one node $x \in \mathcal{V}$ with $v \in \beta(x)$,
2. for each edge $\{u, v\}$ of $E$, there is at least one node $x \in \mathcal{V}$ such that $\beta(x)$ contains $u$ and $v$, and
3. for each vertex $v \in V$, the subgraph $\mathcal{T}[\mathcal{V}_v]$ is connected, where $\mathcal{V}_v := \{x \in \mathcal{V} \mid v \in \beta(x)\}$.

We call $\beta(x)$ the *bag* of $x$. The *width of a tree decomposition* is the size of the largest bag minus one and the *treewidth* of a graph $G$, denoted by $\mathrm{tw}(G)$, is the minimal width of any tree decomposition of $G$.

We consider tree decompositions with specific properties. A node $x \in \mathcal{V}$ is called

1. a *leaf node* if $x$ has no child nodes in $\mathcal{T}$,
2. a *forget node* if $x$ has exactly one child node $y$ in $\mathcal{T}$ and $\beta(y) = \beta(x) \cup \{v\}$ for some $v \in V \setminus \beta(x)$,
3. an *introduce node* if $x$ has exactly one child node $y$ in $\mathcal{T}$ and $\beta(y) = \beta(x) \setminus \{v\}$ for some $v \in V \setminus \beta(y)$, or
4. a *join node* if $x$ has exactly two child nodes $y$ and $z$ in $\mathcal{T}$ and $\beta(x) = \beta(y) = \beta(z)$.

A tree decomposition $(\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*), \beta)$ is called *nice* if the bag of the root and the bags of all leaf nodes are empty sets and if every node $x \in \mathcal{V}$ is either a leaf node, a forget node, an introduce node, or a join node.

For a node $x \in \mathcal{V}$, we denote with $V_x$ the union of all bags $\beta(y)$, where $y$ is contained in the subtree of $\mathcal{T}$ rooted in $x$. Moreover, we denote $G_x := G[V_x]$ and $E_x := E_G(V_x)$.

To obtain small polynomial factors in the running time, we first analyze the number of subsets of size at most $k$ of any set $X$ of size $x$. We denote by $\binom{x}{\leq k}$ denote the number of different subsets of $X$ that have size at most $k$, that is, $\binom{x}{\leq k} := \sum_{r=0}^{\min(k,x)} \binom{x}{r}$.

**Lemma 3.14.** *Let $k \geq 1$ be an integer and let $X$ be an arbitrary set of size $x \geq 3$. Then, $\binom{x}{\leq k} \leq 256 \cdot (x-1)^k/k^2$.*

*Proof.* Note that $\binom{x}{\leq k} = \sum_{r=0}^{k} \binom{x}{r} \leq 2^x$ and $\binom{x}{\leq k} \leq x^k$. First, if $k \leq 4$, then $x^k \leq (2 \cdot (x-1))^k \leq 16 \cdot (x-1)^k$ and since $k^2 \leq 16$, $\binom{x}{\leq k} \leq 256 \cdot (x-1)^k/k^2$. Second, if $k \geq \max\{4, x/2\}$, then $4^k \geq 2^x$ and $2^k \geq k^2$. Hence, if $x \geq 9$, then by the fact that $k \geq x/2$ and $x \geq 3$, we get

$$\binom{x}{\leq k} \leq 2^x \leq 4^k \leq \frac{8^k}{2^k} \leq \frac{8^k}{k^2} \leq \frac{(x-1)^k}{k^2}.$$

If $4 \leq x \leq 8$, then $\binom{x}{\leq k} \leq 2^x \leq 256 \cdot (x-1)^k/k^2$ since $(x-1)^k > k^2$ for all $k \geq 2$, and for $x = 3$, $\binom{x}{\leq k} \leq 2^x = 8 \leq 256 \cdot 2^k/k^2$ for all $k \geq 2$. Finally, if $4 < k < x/2$, then $2 \cdot \binom{x}{k} \geq \sum_{r=0}^{k} \binom{x}{r} = \binom{x}{\leq k}$. Hence,

$$\begin{aligned}\binom{x}{\leq k} &\leq 2 \cdot \binom{x}{k} \leq 2 \cdot \frac{x!}{(x-k)! \cdot k!} \leq 2 \cdot x \cdot \frac{(x-1)!}{(x-k)! \cdot k^2} \\ &\leq 2 \cdot 2(x-1) \cdot \frac{(x-1)^{k-1}}{k^2} = 4\frac{(x-1)^k}{k^2} < 256 \cdot \frac{(x-1)^k}{k^2}.\end{aligned}$$

This completes the proof. □

Intuitively, the algorithms based on tree decompositions are obtained by a dynamic programming algorithm on a given tree decomposition of width $r$, where each entry of the dynamic programming table considers the intersection of the current bag of size at most $r+1$ with an improving swap $W$ of size at most $k$.

**Theorem 3.15.** *Let $G = (V, E)$ be an undirected graph, let $\omega \colon V \to \mathbb{N}$ be a weight function, let $S \subseteq V$ be a vertex cover of $G$, let $k$ be a natural number, and let $(\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*), \beta)$ be a nice tree decomposition of width $r$ for $G$ with $\mathcal{O}(r \cdot n)$ nodes. One can compute in $\mathcal{O}((r^{k+1} + k^2) \cdot n)$ time a valid $k$-swap $W$ for $S$ in $G$ such that $\delta(W)$ is maximal under all valid $k$-swaps for $S$ in $G$.*

*Proof.* Due to Lemma 3.9, the statement holds for $k \leq 1$. In the following, we show the running time by describing a dynamic programming algorithm for $k \geq 2$.

Let $N_x(U) := N(U) \cap \beta(x)$ denote the neighbors of $U$ in the bag of $x \in \mathcal{V}$. Recall that for a node $x \in \mathcal{V}$, the vertex set $V_x$ is the union of all bags $\beta(y)$, where $y$ is a node of the subtree of $\mathcal{T}$ rooted in $x$. Moreover, recall that for a node $x \in \mathcal{V}$, $G_x := G[V_x]$ and $E_x := E_G(V_x)$, where $V_x$ is the union of all bags $\beta(y)$, where $y$ is contained in the subtree of $\mathcal{T}$ rooted in $x$.

For each node $x \in \mathcal{V}$ in the tree decomposition, the dynamic programming table $D_x$ has entries of type $D_x[S_x, C_x, k']$ with, $S_x \subseteq S \cap \beta(x)$, $C_x \subseteq \beta(x) \setminus (N(S_x) \cup S)$ and $k' \in [0, k]$, such that $|W_x| \leq k'$ where $W_x := S_x \cup C_x \cup (N_x(S_x) \setminus S)$.

Each entry stores the maximal improvement $\delta_S(W)$ of a valid $k'$-swap $W \subseteq V_x$ for $S \cap V_x$ in $G_x$ such that $W \cap S \cap \beta(x) = S_x$ and $W \cap \beta(x) \setminus (N(S_x) \cup S) = C_x$. In other words, $W$ intersects with the vertices of $S$ of the current bag exactly in $S_x$ and $W$ intersects with the vertices of $V \setminus S$ of the current bag (minus the vertices that are contained in the extension of $S_x$) exactly in $C_x$. Since we restrict $W$ to be a $k$-swap, the size of $S_x \cup C_x$ is thus upper-bounded by $k$ for each reasonable choice of $S_x$ and $C_x$. Moreover, since both these subsets are disjoint, this implies that there are at most $|\beta(x)|^k$ choices for $S_x \cup C_x$ one has to consider in the dynamic programming table.

Since for $|S_x \cup C_x| > k$, $D_x[S_x, C_x, k']$ is not an entry of the dynamic programming table, we define a function $f_x$ to prevent the evaluation of non-existing table entries. For each nodes $x \in \mathcal{V}$, each subset $S_x \subseteq \beta(x) \cap S$, each subset $C_x \subseteq \beta(x) \setminus S$, and each $k' \in [0, k]$, we set $f_x(S_x, C_x, k') := D_x[S_x, C_x, k']$ if $|S_x \cup C_x| \leq k$ and $f_x(S_x, C_x, k') := -\infty$, otherwise.

Next, we describe how to we compute the entries of the dynamic programming tables. For each leaf node $\ell$ of $\mathcal{T}$, we fill the table $D_\ell$ by setting $D_\ell[\emptyset, \emptyset, k'] := 0$ for each $k' \in [0, k]$. This is correct, since $G_\ell$ is the empty graph.

For all non-leaf nodes $x$ of $\mathcal{T}$, we set $D_x[S_x, C_x, k'] := -\infty$ if

- $S_x$ is not an independent set in $G$,
- $|S_x \cup C_x \cup (N_x(S_x) \setminus S)| > k'$, or
- $N(S_x) \cap C_x \neq \emptyset$.

Note that this is correct since in all three cases, there is no swap fulfilling the constraints of the table definition. To compute the remaining entries $D_x[S_x, C_x, k']$, we distinguish between the three types of non-leaf nodes. For each type, we omit the formal proof and only give an informal argument for of the correctness.

**Forget nodes.** Let $x$ be a forget node, let $y$ be the unique child of $x$ in $\mathcal{T}$, and let $v$ be the unique vertex in $\beta(y) \setminus \beta(x)$. The entries for $x$ can be computed as follows:

$$D_x[S_x, C_x, k'] := \begin{cases} \max(f_y(S_x, C_x, k'), f_y(S_x \cup \{v\}, C_x \setminus N(v), k')) & v \in S \text{ and} \\ \max(f_y(S_x, C_x, k'), f_y(S_x, C_x \cup \{v\}, k')) & v \notin S. \end{cases}$$

Informally, we chose the larger improvement of the best swap containing $v$ and the best swap not containing $v$. To consider the best swap containing $v$, we add $v$ to the corresponding set ($S_x$ or $C_x$) for the entry of the table $D_x$. If $v$ is a vertex of $S$, we also have to remove the vertices of $N(v) \setminus S$ from $C_x$, since these vertices are implicitly stored in the corresponding entry of $D_y$ and, by definition, $N(S_y) \cap C_y = \emptyset$.

**Introduce nodes.** Let $x$ be an introduce node, let $y$ be the unique child of $x$ in $\mathcal{T}$, and let $v$ be the unique vertex in $\beta(x) \setminus \beta(y)$. For $W_x := S_x \cup C_x \cup (N_x(S_x) \setminus S)$ and $C^* := (N_x(v) \setminus S) \setminus N(S_x \setminus \{v\})$, the entries for $x$ can be computed as follows:

$$D_x[S_x, C_x, k'] := \begin{cases} f_y(S_x \setminus \{v\}, C_x \cup C^*, k' - 1) + \omega(v) & v \in W_x \cap S, \\ f_y(S_x, C_x \setminus \{v\}, k' - 1) - \omega(v) & v \in W_x \setminus S, \\ f_y(S_x, C_x, k') & \text{otherwise}, \end{cases}.$$

Informally, if $v$ is a vertex of $W_x$, we have to consider the entry of $D_y$ where $v$ is removed from the corresponding set ($S_x$ or $C_x$) and adding the improvement we obtain from having $v$ in the considered swap (increasing by $\omega(v)$ if $v \in S$ and decreasing by $\omega(v)$ if $v \notin S$). If $v$ is a vertex of $S$, the vertices of $C^*$ are not stored implicitly in $D_y$, so we have to consider the entry of $D_y$ where we also explicitly swap $C^*$. Otherwise, if $v$ is not a vertex of $W_x$, we consider the entry of $D_y$ with the same subsets $S_x$ and $C_x$ and the same budget $k'$.

**Join nodes.** Let $x$ be a join node, let $y$ and $z$ be the unique children of $x$ in $\mathcal{T}$. Recall that $\beta(x) = \beta(y) = \beta(z)$. For $W_x := S_x \cup C_x \cup (N_x(S_x) \setminus S)$, the entries for $x$ can be computed as follows:

$$D_x[S_x, C_x, k'] := \max_{0 \leq k'' \leq k' - |W_x|} D_y[S_x, C_x, k'' + |W_x|] + D_z[S_x, C_x, k' - k''] - \delta(W_x).$$

Informally, we divide the budget $k'$ into two parts. One for the subset of vertices of $W$ contained in the subtree rooted in $y$ and one for the subset of vertices of $W$ contained in the subtree rooted in $z$. Note that $W_x$ is contained in both of these

vertex sets. Hence, we consider all possible ways to divide $k'$ into two parts, such that both entries have at least enough budget to swap all vertices of $W_x$. Since the improvement of $W_x$ is added twice, we have to remove $\delta(W_x)$ from the obtained sum.

The maximal improvement of any valid $k$-swap for $S$ in $G$ can then be found in $D_{x^*}[\emptyset, \emptyset, k]$. Moreover, a corresponding swap $W^*$ can be found via traceback: $W^* \cap S$ consists of those vertices that are added to the set $S_x$ in introduce nodes $x$, and $W^* \setminus S$ is exactly $N(W^* \setminus S) \setminus S$.

It remains to show the running time. Recall that $(\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*), \beta)$ is a nice tree decomposition of width $r$ for $G$ with $\mathcal{O}(r \cdot n)$ nodes. The number of entries of the table $D_x$ is upper bounded by $k+1$ times the number of subsets of $\beta(x)$ of size at most $k$. Since for each node $x \in \mathcal{V}$, the bag $\beta(x)$ has size at most $r+1$, all dynamic programming tables together contain $\mathcal{O}(\binom{r+1}{\leq k} \cdot k \cdot r \cdot n)$ entries. Recall that $\binom{r+1}{\leq k}$ denotes the number of different subsets of size at most $k$ of a set of size $r+1$. To complete the proof, it is sufficient to show, that we can compute each of them in $\mathcal{O}(k)$ time.

To this end, we do the following preprocessing in which we assume that all considered subsets of $V$ are stored as sorted lists.

First, we compute a degeneracy ordering $\sigma$ of $G$ in $\mathcal{O}(n+m) \subseteq \mathcal{O}(n \cdot r)$ time. Note that the degeneracy of $G$ is never larger than the treewidth of $G$. Hence, with the help of $\sigma$, we can check in $\mathcal{O}(r)$ time whether two given vertices are adjacent. Next, we compute the adjacency matrix $A(x)$ of $G[\beta(x)]$ for each node $x \in \mathcal{V}$. Since $\beta(x^*) = \emptyset$ for the root vertex $x^*$, we start with an empty adjacency matrix. We now show that for each node $x \in \mathcal{V}$ where $A(x)$ is already computed, we can compute $A(y)$ for each child node $y$ of $x$ in $\mathcal{O}(r^2)$ time. Let $x$ be a forget node with the unique child $y$ and let $v$ be the unique vertex of $\beta(y) \setminus \beta(x)$. Then, we can copy $A(x)$ in $\mathcal{O}(r^2)$ time and add a new row and a new column for the adjacency of $v$. To fill the new column and row, we only have to evaluate for each vertex $w \in \beta(x)$, if $v$ and $w$ are adjacent. These are at most $r$ evaluations running in $\mathcal{O}(r)$ time each. Let $x$ be an introduce node with the unique child $y$ and let $v$ be the unique vertex in $\beta(x) \setminus \beta(y)$. Then, we obtain $A(y)$ by copying $A(x)$ and removing the row and the column of $v$ in $\mathcal{O}(r^2)$ time. Let $x$ be a join node with the unique children $y$ and $z$, then $A(x) = A(y) = A(z)$ and we can obtain these copies of $A(x)$ in $\mathcal{O}(r^2)$ time. Recall that $\beta(x^*) = \emptyset$ for the root $x^*$. Hence, $A(x^*)$ is the empty matrix which can be computed in $\mathcal{O}(1)$ time. Since $\mathcal{T}$ contains $\mathcal{O}(r \cdot n)$ nodes, we can compute $A(x)$ for all nodes $x \in \mathcal{V}$ in $\mathcal{O}(r^3 \cdot n)$ time. Since $k > 2$, this can be upper-bounded by $\mathcal{O}(r^k \cdot n)$ time. Hence, in the following, we can check for each node $x \in \mathcal{V}$ in $\mathcal{O}(1)$ time whether two given vertices of $\beta(x)$ are adjacent. Further, for each node $x \in \mathcal{V}$, we perform the following steps:

- We store all independent sets $S_x \subseteq \beta(x) \cap S$ of size at most $k$. This preprocess-

ing runs in $\mathcal{O}(\binom{r+1}{\leq k} \cdot k^2)$ time, since for each subset $S_x \subseteq \beta(x) \cap S$, we can check in $\mathcal{O}(k^2)$ time, whether $S_x$ is an independent set, by the fact that $|S_x| \leq k$.

- For each independent set $S_x \subseteq \beta(x) \cap S$ of size at most $k$, we store the neighborhood $N_x(S_x) \setminus S$ if it has size at most $k$. Otherwise, we store $\perp$. This preprocessing runs in $\mathcal{O}(\binom{r+1}{\leq k} \cdot k^2)$ time: First consider all subset $S_x$ of size at most $k-1$, this can be done in $\mathcal{O}(r \cdot k)$ time since $|\beta(x)| \leq r+1$ and $|S_x| \leq k-1$. Second, consider all subset $S_x$ of size exactly $k$. Choose an arbitrary vertex $u \in S_x$. Note that $N_x(S_x) \setminus S = (N_x(S_x \setminus \{u\}) \setminus S) \cup (N_x(u) \setminus S)$. Since these two subsets are already stored and have size at most $k$ (or are set to $\perp$) and the union of two sorted lists can be computed in linear time, for each such subset $S_x$, this can be done in $\mathcal{O}(k)$ time. Hence, the total running time is $\mathcal{O}(\binom{r+1}{\leq k} \cdot k)$ time.

- For each independent set $S_x \subseteq \beta(x) \cap S$ of size at most $k$ and each $C_x \subseteq \beta(x) \setminus S$ with $|S_x \cup C_x| \leq k$, we store the information whether $N(S_x) \cap C_x = \emptyset$. Afterwards, we store the set $W_x := S_x \cup C_x \cup (N_x(S_x) \setminus S)$ if it has size at most $k$. Otherwise, we store $\perp$. For each combination of sets $S_x$ and $C_x$, this preprocessing can be done in $\mathcal{O}(k)$ time: Since $|S_x \cup C_x| \leq k$, the set $N_x(S_x) \setminus S$ is already stored. If $|N_x(S_x) \setminus S| > k$, we immediately set $W_x$ to $\perp$. Otherwise, $W_x$ is the union of three sets of size $\mathcal{O}(k)$ which are given as sorted lists.

  Since there are at most $\binom{r+1}{\leq k}$ combinations of sets $S_x$ and $C_x$, this whole preprocessing runs in $\mathcal{O}(\binom{r+1}{\leq k} \cdot k)$ time.

- Let $x$ be an introduce node with the unique child node $y$ and the unique vertex $v \in \beta(x) \setminus \beta(y)$. Then, for each independent set $S_x \subseteq \beta(x) \cap S$ of size at most $k$ with $|N_x(S_x) \setminus S| \leq k$, we store the set $C^* := (N_x(v) \setminus S) \setminus N(S_x \setminus \{v\})$. Now $C^* = (N_x(v) \setminus S) \setminus N(S_x \setminus \{v\}) = (N_x(v) \setminus S) \setminus (N_x(S_x) \setminus S)$ and the two subsets of the latter expression are already stored and have size at most $k$. Thus, $C^*$ can be computed in $\mathcal{O}(k)$ time for each independent set $S_x$ of size at most $k$, since the difference of two sorted lists can be computed in linear time.

  Moreover, note that $|C^*| \leq k$. Hence, this preprocessing runs in $\mathcal{O}(\binom{r+1}{\leq k} \cdot k)$ time.

Since there are $\mathcal{O}(r \cdot n)$ nodes, the whole preprocessing (including constructing the degeneracy ordering and building all the individual adjacency matrices) runs in $\mathcal{O}((\binom{r+1}{\leq k} \cdot k^2 + \binom{r+1}{\leq k-1} \cdot r^2 \cdot k) \cdot n) = \mathcal{O}((\binom{r+1}{\leq k} \cdot r \cdot k^2) \cdot n)$ time.

Note that with this preprocessing, each entry of the tables $D_x$ can be computed in $\mathcal{O}(k)$ time: For each forget or introduce node $x$, one considers $O(1)$ cases, where for each case, one has to (i) compute set operations for $\mathcal{O}(1)$ sets of size at most $k$ each,

as well as (ii) evaluating one function call the function $f_x(S_x, C_x, k')$. Here, the latter can be done in $\mathcal{O}(k)$ time, since $f_x(S_x, C_x, k')$ only checks whether $|S_x \cup C_x| \leq k$. For each join node $x$, the corresponding set $W_x$ can be computed in $\mathcal{O}(k)$ time and one computes the maximum of $\mathcal{O}(k)$ cases in which we combine two other table entries in $\mathcal{O}(1)$ time. Moreover, note that all dynamic programming tables have $\mathcal{O}((\binom{r+1}{\leq k} \cdot r \cdot k \cdot n)$ entries in total. Since $k > 2$ and due to Lemma 3.14, we thus obtain that the whole algorithm runs in $\mathcal{O}(r^{k+1} \cdot n)$ time if $r \geq 2$, and in $\mathcal{O}(2^r \cdot k^2 \cdot n) = \mathcal{O}(k^2 \cdot n)$ time, otherwise. □

Note that the running time of this algorithm can also be bounded by $\mathcal{O}(2^r \cdot k^2 \cdot n)$ which implies that GLS WVC can be solved in polynomial time on graphs with a constant treewidth. This latter bound is, however, not useful in practice, since one can find some optimal weighted vertex cover in the same running time [35, 131].

The dynamic programming algorithm deviates from the simple idea mentioned above in the following detail: it considers only (i) the intersection $W_x^S$ of $W \cap S$ with the vertices of the current bag and (ii) the intersection of $W$ with those vertices of $V \setminus S$ in the current bag that are not contained in $N(W_x^S)$. This is more technical but has the following benefit: The intersection of $W$ with $N(S_x) \setminus S$ is stored implicitly which decreases the factor $r^{k+1}$ to $r^{\frac{k+d}{2}+1}$ for GLS VC due to Lemma 3.7. In particular, for the case of $d = 1$, that is, for LS VC, this gives a substantial improvement of the exponential part of the running time from $r^{k+1}$ to $r^{\frac{k+1}{2}+1}$.

**Theorem 3.16.** *Let $I = (G = (V, E), S, k, d)$ be an instance of* GLS VC. *When given a nice tree decomposition of width $r$ for $G$ with $\mathcal{O}(r \cdot n)$ nodes, one can solve $I$ in $\mathcal{O}((r^{\frac{k+d}{2}+1} + k^2) \cdot n)$ time.*

*Proof.* Due to Lemma 3.10, one can solve $I$ in $\mathcal{O}(n+m) \subseteq \mathcal{O}(n \cdot r)$ time if $k+d \leq 4$. In the following, we may thus assume $\frac{k+d}{2} > 2$. To obtain the stated running time, we modify the dynamic program described in the proof of Theorem 3.15. We limit the entries of the dynamic programming table such that $|S_x \cup C_x| \leq \frac{k+d}{2}$. Hence, we also update $f_x(S_x, C_x, k')$ to $-\infty$ if $|S_x \cup C_x| > \frac{k+d}{2}$. To determine if there is a valid $d$-improving $k$-swap for $S$ in $G$ it is thus sufficient to check whether $D_{x^*}[\emptyset, \emptyset, k]$ is at least $d$, where $x^*$ denotes the root of $\mathcal{T}$. Moreover, the corresponding swap can be found via traceback.

The correctness of this modified dynamic program relies on Lemma 3.7 which, intuitively speaking, states that, if there is a good swap for $I$, then such a swap can be found by only considering entries of the dynamic programming table where $|S_x \cup C_x| \leq \frac{k+d}{2}$.

Since $\frac{k+d}{2} \geq 2$, the adjacency matrix of $G[\beta(x)]$ can be computed in $\mathcal{O}(r^{\frac{k+d}{2}+1} \cdot n)$ time for all $x \in \mathcal{V}$. Moreover, since we now only check for subsets of size at most $\frac{k+d}{2}$, the preprocessing, filling all entries of the table, and checking $D_{x^*}[\emptyset, \emptyset, k] \geq d$ can be done in $\mathcal{O}(\binom{r+1}{\leq \frac{k+d}{2}} \cdot r \cdot k^2 \cdot n)$ time which, due to Lemma 3.14, is $\mathcal{O}(r^{\frac{k+d}{2}+1} \cdot n)$ time for $r \geq 2$ and $\mathcal{O}(k^2 \cdot n + m) \subseteq \mathcal{O}(k^2 \cdot n \cdot r)$ time, otherwise. □

Since computing a tree decomposition of minimal width is NP-hard, we cannot directly obtain a running time of $\mathcal{O}((\mathrm{tw}(G)^{k+1}+k^2)\cdot n)$ and $\mathcal{O}((\mathrm{tw}(G)^{\frac{k+d}{2}+1}+k^2)\cdot n)$, respectively. We can, however, compute a nice tree decomposition of width $\mathrm{tw}(G)$ in $\mathcal{O}(n+m)$ time if $\mathrm{tw}(G) \leq 1800$ [21]. Moreover, for each $r \geq 0$, one can compute a nice tree decomposition of $G$ of width at most $1800 \cdot r^2$ or correctly output that $\mathrm{tw}(G) > r$ in $\mathcal{O}(r^7 \cdot n \cdot \log(n))$ time [60]. Hence, we can compute in $\mathcal{O}(\mathrm{tw}(G)^8 \cdot n \cdot \log(n))$ time [60] a nice tree decomposition of $G$ of width at most $1800 \cdot \mathrm{tw}(G)^2$ by applying the approximation algorithm for each $r \in [1, \mathrm{tw}(G)]$. If $\mathrm{tw}(G) \geq 1800$, then the width of the latter tree decomposition is smaller than $\mathrm{tw}(G)^3$. Altogether, we obtain the following by Lemma 3.10 and Lemma 3.9.

**Corollary 3.17.** GLS VC *can be solved in* $\mathcal{O}((\mathrm{tw}(G)^{\frac{3\cdot(k+d)}{2}+1}+k^2)\cdot n\cdot\log(n))$ *time and* GLS WVC *can be solved in* $\mathcal{O}((\mathrm{tw}(G)^{3k+1}+k^2)\cdot n\cdot\log(n))$ *time.*

Note that even if a tree decomposition of width $\mathrm{tw}(G)$ is given, one cannot improve much on the running time due to the lower bound of Theorem 3.11, since the treewidth of a graph is never larger than its vertex cover number. Due to this relation, we further obtain the following by initially computing a 2-approximating of a minimum vertex cover.

**Corollary 3.18.** *For a given vertex cover* $S^*$ *of* $G$, GLS VC *can be solved in* $\mathcal{O}(|S^*|^{\frac{(k+d)}{2}+1}\cdot n+m)$ *time and* GLS WVC *can be solved in* $\mathcal{O}(|S^*|^{k+1}\cdot n+m)$ *time. In general,* GLS VC *can be solved in* $\mathcal{O}((2\cdot \mathrm{vc}(G))^{\frac{(k+d)}{2}}\cdot n+m)$ *time and* GLS WVC *can be solved in* $\mathcal{O}((2\cdot \mathrm{vc}(G))^k\cdot n+m)$ *time.*

Since $S$ is a vertex cover of $G$, this also implies an algorithm for GLS VC with running time $\mathcal{O}(|S|^{\frac{(k+d)}{2}+1}\cdot n+m)$ and an algorithm for GLS WVC with running time $\mathcal{O}(|S|^{k+1}\cdot n+m)$. In particular, $S^*$ may be the current vertex cover $S$.

## 3.3 Degree-Related Parameterizations

In this section, we present FPT-algorithms with running times that grow strongly with respect to $k$ and grow only mildly with respect to $\Delta(G)$ or the $h$-index of $G$.

In contrast to previous work, these FPT-algorithms solve the more general problems with weights and gap-improvements; the algorithms for $\Delta(G)$ will be used as subroutines in the algorithm for the $h$-index of $G$.

We start by presenting an algorithm for instances with an $h$-index of at most 1 which will be used to handle border cases for both parameterizations.

**Lemma 3.19.** GLS WVC *can be solved in* $\mathcal{O}(k \cdot \log(k) + n)$ *time if the* $h$*-index of* $G$ *is at most* 1.

*Proof.* Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS WVC with $h(G) \leq 1$. Note that this implies that $m \in \mathcal{O}(n)$.

First, we present an algorithm with the stated running time if $\Delta(G) \leq 1$. To this end, we apply an initial reduction rule. For each edge $e := \{u, v\} \in E$ where $S$ contains both $u$ and $v$, we remove vertex $v$ from $G$ if $\omega(v) \leq \omega(u)$. Otherwise, we remove vertex $u$ from $G$. Since each vertex cover of $G$ contains at least one endpoint of $e$, it is never optimal to swap the endpoint of $\{u, v\}$ of smaller weight out of $S$ and thus, the reduction rule produces an equivalent instance. Moreover, note that this reduction rule can be applied simultaneously to all edges of $E$ where both endpoints are in $S$. This implies that this reduction rule can be applied exhaustively in $\mathcal{O}(n)$ time.

Hence, in the following, we can assume that $S$ contains exactly one endpoint of each edge of $E$. Since $\Delta(G) \leq 1$, each valid connected swap consists either of (i) a degree-0 vertex in $S$ or (ii) both endpoints of an edge.

We now compute $\mathcal{W}_1 := \{\{v\} \mid v \in S, N(v) = \emptyset\}$ the set of valid 1-swaps and $\mathcal{W}_2 := \{\{v, w\} \mid \{v, w\} \in E, \{v, w\} \not\subseteq S, \delta(\{v, w\}) > 0\}$ the set of valid connected improving 2-swaps for $S$ in $G$ in $\mathcal{O}(n)$ time. Note that each vertex is contained in at most one set in $\mathcal{W}_1 \cup \mathcal{W}_2$. Since $\Delta(G) \leq 1$, the union $W$ of each subset of swaps of $\mathcal{W}_1 \cup \mathcal{W}_2$ is a valid swap for $S$ in $G$. Hence, it remains to find some set $W$ of size at most $k$ with the maximal improvement. To this end, we compute the set $\mathcal{W}_i^k$ of the $k$ elements of $\mathcal{W}_i$ of maximal improvement and sort them according to their weight for each $i \in \{1, 2\}$. This can be done by finding the swap $W_i^k$ in $\mathcal{W}_i$ with the $k$-th largest improvement by using the median of the medians in $\mathcal{O}(n)$ time and afterwards filtering all swaps of improvement at least $\delta(W_i^k)$ and sorting them in $\mathcal{O}(k \cdot \log(k) + n)$ time. Next, we start with an empty set $W$. If $k = 0$, $I$ is a yes-instance of GLS WVC if $\delta(W) \geq d$. If $k = 1$, update $W$ to $W \cup W_1$ and $k$ to 0, where $W_1$ is the swap in $\mathcal{W}_1^k$ with the maximal improvement. If $k \geq 2$, let $W_1$ and $W_1'$ be the two swaps in $\mathcal{W}_1^k$ with the maximal improvement and let $W_2$ be the swap in $\mathcal{W}_2^k$ with the maximal improvement. Then, update $W$ to $W \cup W_1 \cup W_1'$ and remove both $W_1$ and $W_1'$ from $\mathcal{W}_1^k$ if $\delta(W_1 \cup W_1') \geq \delta(W_2)$. Otherwise, update

to $W \cup W_2$ and remove $W_2$ from $\mathcal{W}_2^k$. In both cases reduce $k$ by two. This greedy algorithm is correct and runs in $\mathcal{O}(k \cdot \log(k) + n)$ time.

Next, we analyze the case where $\Delta(G) > 1$. Since $h(G) \leq 1$, there is exactly one vertex $v^*$ of degree at least 2 in $G$. First, we compute the swap-instance $I_1 := (G_1, \omega_1, S_1, k_1, d_1) := \text{swap}(I, \{v_x\})$. Then, since $G_1$ has a maximum degree of at most one, we can check whether $I_1$ is a yes-instance of GLS WVC in $\mathcal{O}(k \cdot \log(k) + n)$ time by using the above algorithm. If $I_1$ is a yes-instance of GLS WVC, then $I$ is a yes-instance of GLS WVC. Otherwise, $v_x$ is contained in no valid $d$-improving $k$-swap for $S$ in $G$. Hence, let $I_2 := (G_2, \omega_2, S_2, k, d)$ be the instance of GLS WVC we obtain by removing $v_x$ and, if $v_x \in V \setminus S$, all vertices in $N(v_x) \cap S$ from $G$. Since $G_2$ has a maximum degree of one we can solve the instance $I_2$ in $\mathcal{O}(k \cdot \log(k) + n)$ time. Moreover, since $I$ is a yes-instance of GLS WVC if and only if $I_2$ is a yes-instance of GLS WVC, we obtain the stated running time of $\mathcal{O}(k \cdot \log(k) + n)$. □

Hence, for the following algorithms, we assume that $h(G)$ and $\Delta(G)$ are at least 2.

### 3.3.1 Parameterizing Unweighted Gap Local Search by the Maximum Degree

In this section, we present an algorithm for GLS VC that runs mildly with respect to $\Delta(G)$ and strongly with respect to $k+d$. Recall that LS VC (the version of GLS VC with $d = 1$) can be solved in $2^k \cdot (\Delta(G) - 1)^{\frac{k+1}{2}} \cdot n^{\mathcal{O}(1)}$ time. Except for a change from the $2^k$-factor to a $k!$-factor, the algorithm we present essentially directly generalizes this running time for larger values of $k$.

**Theorem 3.20.** GLS VC *can be solved in* $\mathcal{O}(k! \cdot (\Delta - 1)^{\frac{k+d}{2}} \cdot n)$ *time.*

The first idea for an algorithm is obviously to adapt the known $\mathcal{O}(2^k \cdot (\Delta - 1)^{\frac{k+1}{2}} \cdot k^2 \cdot n)$-time algorithm for LS VC [98] to GLS VC. This algorithm, however, relies on the fact that for LS VC, it is sufficient to consider only connected swaps. For $d$-improving swaps this is not the case: for $k = 2$, an improvement of at least 2 can only be achievable by swapping two vertices that out of the current solution that are not adjacent. These vertices may have an arbitrarily large distance in the graph. Thus, the gap version of the problem becomes considerably harder.

To avoid considering all possible vertex sets of size at most $k$, we present two branching rules. The first rule applies if there is a vertex $v$ in $S$ where $N(v) \subseteq S$ and branches in all possible ways to swap either $v$ or two non-adjacent vertices of $N(v)$. If this rule cannot be applied, then each vertex in $S$ has at least one neighbor in $V \setminus S$ and, thus, there is no valid improving swap of size one.

**Proposition 3.21.** *Let $I = (G, S, k, d)$ be a yes-instance of* GLS VC *and let $v$ be a vertex of $S$ with $N(v) \subseteq S$. There is a good swap $W$ for $I$, such that (i) $v$ is contained in $W$ or (ii) $W$ contains at least two neighbors of $v$.*

*Proof.* Let $W$ be a good swap for $I$. Suppose that $v$ is not contained in $W$ and that $W$ contains at most one neighbor of $v$, as otherwise the statement already holds. If $W$ contains no neighbor of $v$, let $w$ be an arbitrary vertex of $S \cap W$. Otherwise, let $w$ be the unique vertex in $W \cap N(v)$. Note that $w \in S \cap W$ by the fact that $N(v) \subseteq S$. Hence, $W' := W \setminus \{w\}$ is a valid $(d-1)$-improving $(k-1)$-swap for $I$. Observe that $W'$ contains neither $v$ nor a neighbor of $v$. Consequently, $W^* := W' \cup \{v\}$ is a valid $d$-improving $k$-swap for $S$ in $G$. Since $W^*$ contains $v$, the statement holds. □

From Proposition 3.21, we derive the following branching rule. Here, we swap $v$ or two independent neighbors of $v$ in each case. This gives a better branching vector than swapping only a single vertex of $N[v]$ in each case.

**Branching Rule 3.3.1.** Let $I = (G = (V, E), S, k, d)$ be an instance of GLS VC and let $v$ be a vertex of $S$ with $N(v) \subseteq S$. For each swap $W \in (\binom{N(v)}{2} \setminus E) \cup \{\{v\}\}$, branch into the case of swapping $W$.

As mentioned above, if the branching rule cannot be applied anymore, then each valid improving swap contains at least two vertices. Before applying the second branching rule, we perform the following preprocessing. First, we compute for each $j \in [2, d]$ some minimum valid connected $j$-improving $(k-d+j)$-swap $W_j$ for $S$ in $G$ if there is any. We call a collection $\{W_j\}_2^d$ of such swaps a *swap family*. The idea of the branching rule is the following: for each $\ell \in [2, d]$, either the $d$-improving swap contains $W_\ell$, some neighbor of $W_\ell$, or no connected component that is exactly $\ell$-improving. Intuitively, this is a correct by the following observation: Consider some valid minimum good swap $W$ for $I$ and let $C$ be a connected component in $G[W]$ with the minimal improvement. Let $\ell := \delta(C)$ and let $W' = (W \setminus C) \cup W_\ell$. Since $W_\ell$ contains at most $|C|$ vertices, we have $|W'| \leq k$. The resulting swap $W'$ is a good swap for $I$ if $W \cap N[W_\ell] = \emptyset$.

We now formally prove that this intuition is correct.

**Proposition 3.22.** *Let $I = (G = (V, E), S, k, d)$ be a yes-instance of* GLS VC *and, let $\{W_j\}_2^d$ be a swap family. There is a good swap $W$ for $I$ such that* (i) *$W$ is connected or* (ii) *there is some $j \in [1, \lfloor \frac{d}{2} \rfloor]$ such that $W_j \subseteq W$ or $W \cap N(W_j) \neq \emptyset$.*

*Proof.* Let $W$ be a minimum good swap for $I$. Hence, the improvement of $W$ is exactly $d$, as otherwise removing an arbitrary vertex of $S$ from $W$ yields a good swap as well. Suppose that $W$ is not connected and that for each $j \in [1, \lfloor \frac{d}{2} \rfloor]$, $W_j \not\subseteq W$,

and $W \cap N(W_j) = \emptyset$, as otherwise the statement already holds. Let $C$ be a connected component in $G[W]$ that minimizes $\delta(C)$ among all connected components of $G[W]$, that is, $C$ is the connected swap of $W$ with the smallest improvement. Let $\ell := \delta(C)$. Since $W$ is not connected and has improvement exactly $d$, the improvement $\ell$ of $C$ is at most $\lfloor \frac{d}{2} \rfloor$. Note that $\ell \geq 1$, as otherwise $W$ is not minimum.

Note that $W' := W \setminus C$ is a valid $(d-\ell)$-improving $(k-|C|)$-swap for $I$ and $|C| < k$. Since $W$ has size at most $k$ and is $d$-improving, $W \setminus S$ is $(d-\ell)$-improving. This implies that $|C| \leq k - d + \ell$. Recall that $W_\ell$ is some minimum valid connected $\ell$-improving $(k-d+\ell)$-swap for $S$ in $G$. Hence, $|C| \geq |W_\ell|$. In the following, we show that $W' \cup W_\ell = (W \setminus C) \cup W_\ell$ is a good swap for $I$. To this end, we first show that $W$ contains no vertex of $W_\ell$.

Assume towards a contradiction that $W \cap W_\ell$ is nonempty and let $C_W$ be some connected component of $G[W]$ that contains at least one vertex of $W_\ell$. Recall that by assumption $W_\ell \not\subseteq W$ and $N(W_\ell) \cap W = \emptyset$. Hence, $C_W$ is a proper subset of $W_\ell$. Since $W_\ell$ is a smallest valid $\ell$-improving $k$-swap for $S$ in $G$, $\delta(C_W) < \delta(W_\ell) = \ell = \delta(C)$. This contradicts the fact that $C$ is a connected component of $G[W]$ of minimal improvement. Consequently, $W$ contains no vertex of $W_\ell$.

We set $W^* := W' \cup W_\ell$. Note that $W^*$ is a $d$-improving $k$-swap for $S$ in $G$. It remains to show that $W^*$ is valid. Since $W'$ and $W_\ell$ are both valid, it follows that $W^*$ is valid if $W' \cap N(W_\ell \cap S) \cap S = \emptyset$. By assumption, this is the case. □

Proposition 3.21 now directly implies the correctness the following branching rule.

**Branching Rule 3.3.2.** Let $I = (G = (V, E), S, k, d)$ be an instance of GLS VC such that there is no connected good swap for $I$. Moreover, let $\{W_j\}_2^d$ be a given swap family. For each $j \in [1, \lfloor \frac{d}{2} \rfloor]$ where $W_j$ exists and each swap $W \in \{\{w\} \mid w \in N(W_j)\} \cup \{W_j\}$, branch into the case of swapping $W$.

Since this branching relies on knowing a swap family, we next present an algorithm to efficiently find such a swap family using the algorithm of Katzmann and Komusiewicz [98] as a subroutine.

**Proposition 3.23.** *Let $I = (G = (V, E), S, k, d)$ be an instance of* GLS VC. *For all $j \in [1, d]$, one can compute a swap family $\{W_j\}_2^d$ in time $\mathcal{O}(2^k \cdot (\Delta-1)^{(k+d)/2} \cdot k^3 \cdot n)$.*

The proof of Proposition 3.23 relies on the following lemma.

**Lemma 3.24** ([98])**.** *Let $G = (B \cup W, E)$ be a bipartite and connected graph with partite sets $B$ and $W$ where $|B| > |W|$. Then, there is a vertex set $B' \subseteq B$, such that $|B'| = |W| + 1$ and $B' \cup W$ is connected.*

Since there is no published proof for this lemma, we provide one for the sake of completeness.

*Proof.* We prove the statement by showing that there is a vertex $v \in B$ such that $G[(B \setminus \{v\}) \cup W]$ is connected. Recursively applying this argument then implies the desired statement. Let $T = (B \cup W, E')$ be an arbitrary spanning tree of $G$, which is rooted in some vertex $w \in W$. We show that there is a vertex $v \in B$ which is a leaf of $T$, which then implies that $T[(B \setminus \{v\}) \cup W]$ is a spanning tree of $G[(B \setminus \{v\}) \cup W]$. Let $d$ be the depth of $T$ and for each $i \in [1, d]$, let $L_i$ be the vertices of the $i$th level of $T$. Since $G$ is bipartite, $L_i \subseteq B$ if $i$ is odd and $L_i \subseteq W$ if $i$ is even. If $d$ is odd, then each vertex of $L_d \subseteq B$ is a leaf of $T$. Otherwise, assume $d$ is even. For each odd $i \in [1, d]$, $L_i$ contains a leaf of $T$ or $|L_{i+1}| \geq |L_i|$. Since

$$\sum_{\text{odd } i \in [1,d]} |L_i| = |B| > |W| > \sum_{\text{even } i \in [1,d]} |L_i|,$$

there is at least one odd $i \in [1, d]$ with $|L_{i+1}| < |L_i|$, which implies that $L_i \subseteq W$ contains a leaf of $T$. □

We are now able to prove Proposition 3.23.

*Proof of Proposition 3.23.* Due to Lemma 3.24, for each valid connected improving swap $W$ for $S$ in $G$, there is an independent set $J \subseteq W \cap S$ such that $W' = W \setminus J$ is a valid connected swap for $S$ in $G$ with $|W' \cap S| = |W' \setminus S| + 1$. Katzmann and Komusiewicz [98] presented an algorithm $\mathcal{A}$ to enumerate all valid connected $(k-d+1)$-swaps $W'$ for $S$ in $G$ with $|W' \cap S| = |W' \setminus S| + 1$ in $\mathcal{O}(2^k \cdot (\Delta(G)-1)^{(k-d)/2} \cdot k^2 \cdot n)$ time. Our algorithm relies on the following idea: Let $j \in [1, d]$ and let $W_j$ be a minimum valid connected $j$-improving $(k-d+j)$-swap for $S$ in $G$. Then, Lemma 3.24 implies that there is an independent set $J_j \subseteq W_j \cap S$ of size $j-1$, such that $W_j \setminus J_j$ is a valid 1-improving $(k-d+1)$-swap for $S$ in $G$. Hence, to find the swap $W_j$, the intuitive idea behind our algorithm is to enumerate all valid connected $(k-d+1)$-swaps $W'$ for $S$ in $G$ with $|W' \cap S| = |W' \setminus S| + 1$ by using algorithm $\mathcal{A}$, and afterwards finding an independent set of size $j-1$ in $(V \cap S) \setminus W'$ that can extend $W'$.

We initialize $W_j$ with $\perp$ for each $j \in [1, d]$. For each swap $W'$ outputted by $\mathcal{A}$, we compute in an auxiliary graph $G'_{W'}$ an independent set $J$ of size at most $d-1$ which has maximum size under this property. Intuitively, the graph $G'_{W'}$ is the subgraph of $G$ induced on exactly those vertices of $S$ that can individually be swapped together with $W'$ to still obtain a valid connected swap. That is, $G'_{W'} := G[\{v \in S \setminus W' \mid N(v) \subseteq S \oplus W', v \in N(W' \setminus S)\}]$. Since

- $W'$ contains at least one vertex of $V \setminus S$,

- $G'_{W'}$ contains no vertex of $W'$, and
- each vertex of $G'_{W'}$ is a neighbor of some vertex of $W' \setminus S$ in $G$,

the graph $G'_{W'}$ has a maximum degree of at most $\Delta - 1$. This implies that we can find the set $J$ in $\mathcal{O}((\Delta-1)^{d-1} \cdot (\Delta-1) \cdot k)$ time by a standard search-tree algorithm, since $G'_{W'}$ contains at most $(\Delta-1) \cdot k$ vertices. Note that for each subset $J'$ of $J$, $W' \cup J'$ is a valid connected $(|J'|+1)$-improving $k$-swap for $S$ in $G$. For each $r \in [0, |J|]$, let $J_r$ be an arbitrary subset of $J$ of size $r$ and update $W_{r+1}$ to $W' \cup J_r$ if $|W' \cup J_r| < |W_{r+1}|$.

Next, we show that the algorithm is correct. Let $j \in [1, d]$ such that a minimum $j$-improving $(k-d+j)$-swap $W_j^*$ for $S$ in $G$ exists. We show that the set $W_j$ computed by our algorithm has size $|W_j^*|$. Since $W_j^*$ is a $j$-improving $(k-d+j)$-swap and $W_j^*$ is minimum, there is an independent set $J_j \subseteq W_j^* \cap S$ of size $j-1$ such that $W' := W_j^* \setminus J_j$ is connected due to Lemma 3.24. Hence, $W'$ is a valid connected $(k-d+1)$-swap for $S$ in $G$. As a consequence, the algorithm $\mathcal{A}$ outputs $W'$. Moreover, since $W_j^*$ is valid, for each vertex $v \in J_j$, $N(v) \subseteq S \oplus W'$. Hence, $J_j$ is an independent set of size $j-1$ in $G'_{W'}$. Thus, when considering $W'$, our algorithm finds some independent set $J'$ of size at least $j-1$ in $G'_{W'}$ and either updates $W_j$ to $W' \cup J'_j$ for some subset $J'_j \subseteq J'$ of size $j-1$ or $W_j$ already has size $|W_j^*|$. Hence, after our algorithm considered the swap $W'$, $|W_j|$ is exactly $|W' \cup J'_j| = |W_j^*|$ since $W^*$ is minimum. Moreover, since $G'_{W'}$ contains only vertices that are adjacent to at least one vertex of $W' \setminus S$, $W' \cup J'_j$ is connected.

Hence, for all $j \in [1, d]$, if it exists, we can find some minimum valid connected $j$-improving $k$-swap $W_j$ for $I$ where $|W_j \setminus S| \leq (k-d)/2$ in total time $\mathcal{O}(2^k \cdot \Delta^{(k+d)/2} \cdot k^3 \cdot n)$. □

With the above two branching rules and the algorithm to compute a swap family, we are now able to prove Theorem 3.20.

*Proof of Theorem 3.20.* Let $I = (G = (V, E), S, k, d)$ be an instance of GLS VC. If $k + d \leq 4$, we can solve $I$ in $\mathcal{O}(n+m)$ time due to Lemma 3.10. If $\Delta(G) = 2$, then we can compute a nice tree decomposition of $G$ of width at most 2 in $\mathcal{O}(n+m)$ time and afterwards solve $I$ in $\mathcal{O}((2^{(k+d)/2} + k^2) \cdot n + m)$ time due to Theorem 3.16. Since $d \leq k$, this is $\mathcal{O}(k! \cdot n + m)$ time. Hence, we can assume in the following, that $k + d > 4$ and $\Delta(G) \geq 3$.

First, check in $\mathcal{O}(n+m)$ time if there is a vertex $v \in S$ with $N(v) \subseteq S$. If this is the case, apply Branching Rule 3.3.1. Due to Proposition 3.21, this is correct.

If there is no vertex $v \in S$ with $N(v) \subseteq S$, compute a swap family $\{W_j\}_2^d$. Due to Proposition 3.23, this can be done in $\mathcal{O}(2^k \cdot (\Delta-1)^{(k+d)/2} \cdot k^3 \cdot n)$ time. If $W_d$

exists, answer yes. Otherwise, apply Branching Rule 3.3.2. Due to Proposition 3.22, this is correct.

It remains to show the total running time. Let $\mathcal{I}_1$ denote the set of swap-instances considered during one application of Branching Rule 3.3.1 and let $\mathcal{I}_2$ denote the set of swap-instances considered during one application of Branching Rule 3.3.2. We first analyze $\mathcal{I}_1$ and $\mathcal{I}_2$. Note that $\mathcal{I}_1$ consists of at most $\binom{\Delta}{2}$ instances where $k' \leq k-2$ and $k'+d' \leq k+d-4$ and exactly one instance where $k' = k-1$ and $k'+d' = k+d-2$. Since $\Delta \geq 3$, $\binom{\Delta}{2} \leq (\Delta-1)^2$.

Hence, if Branching Rule 3.3.1 is applicable, the recurrence for the running time is

$$T(k, k+d) \leq (\Delta-1)^2 \cdot T(k-2, k+d-4) + T(k-1, k+d-2) + \mathcal{O}(n+m).$$

Under the assumption that Branching Rule 3.3.1 is not applicable, we bound the number of instances in $\mathcal{I}_2$ and show that for each such instance $(G', S', k', d') \in \mathcal{I}_2$, $k' \leq k-2$ and $k'+d' \leq k+d-2$. Let $\ell$ be the largest number of $[1, \lfloor \frac{d}{2} \rfloor]$ such that $W_\ell$ exists. Note that $\ell \leq \frac{d}{2} < d \leq k$. Hence, for each $j \in [1, \ell]$, the swap $W_j$ has size at most $k-1$. Further, since $W_j$ is connected, each vertex in $W_j$ has at least one neighbor in $W_j$ and thus at most $\Delta - 1$ neighbors outside of $W_j$. Altogether,

$$|\mathcal{I}_2| \leq \frac{d}{2} \cdot ((k-1) \cdot (\Delta-1) + 1) \leq \frac{k^2}{2} \cdot (\Delta-1),$$

since $\Delta \geq 2$. By the assumption that Branching Rule 3.3.1 is not applicable, there is no vertex $v \in S$ with $N(v) \subseteq S$. This implies that $k' \leq k-2$. Further recall that since each considered swap contains at least one vertex of $S$, $k'+d' \leq k+d-2$ for each instance of $\mathcal{I}_2$.

Hence, if Branching Rule 3.3.1 is not applicable and Branching Rule 3.3.2 is applicable, the recurrence for the running time is

$$T(k, k+d) \leq \frac{k^2}{2} \cdot (\Delta-1) \cdot T(k-2, k+d-2) + \mathcal{O}(2^k \cdot (\Delta-1)^{(k+d)/2} \cdot k^2 \cdot n).$$

We now show by induction over $k$ that there is a constant $C$ such that for each $d \in [1, k]$, $T(k, k+d) \leq C \cdot k! \cdot (\Delta-1)^{(k+d)/2} \cdot n$.

By the above, there is a constant $c$ such that for each $k \in \mathbb{N}$ and each $d \in [1, k]$

**(1)** GLS VC can be solved in $c \cdot (n+m)$ time if $k < 3$,

**(2)** $T(k, k+d) \leq (\Delta-1)^2 \cdot T(k-2, k+d-4) + T(k-1, k+d-2) + c \cdot (n+m)$ if there is some vertex $v \in S$ with $N(v) \subseteq S$, and

**(3)** $T(k, k+d) \leq \frac{k^2}{2} \cdot (\Delta - 1) \cdot T(k-2, k+d-2) + c \cdot 2^k \cdot (\Delta - 1)^{(k+d)/2} \cdot k^3 \cdot n$ if there is no vertex $v \in S$ with $N(v) \subseteq S$.

We set $C := 128c$. Hence, for the base case $k < 3$ the statement holds directly due to **(1)**. As the inductive step, we show that the statement holds for $k$ if it holds for all $k' \in [1, k-1]$.

Suppose that there is some vertex $v \in S$ with $N(v) \subseteq S$. By the induction hypothesis and due to **(2)**,

$$\begin{aligned}
T(k, k+d) &\leq (\Delta-1)^2 \cdot T(k-2, k+d-4) + T(k-1, k+d-2) + c \cdot (n+m) \\
&\leq C \cdot (\Delta-1)^2 \cdot (k-2)! \cdot (\Delta-1)^{(k+d)/2-2} \cdot n \\
&\quad + \; C \cdot (k-1)! \cdot (\Delta-1)^{(k+d)/2-1} \cdot n + c \cdot (n+m) \\
&= C \cdot (k-2)! \cdot (\Delta-1)^{(k+d)/2-1} \cdot ((\Delta-1) + (k-1)) \cdot n + c \cdot (n+m) \\
&\overset{(*)}{\leq} C \cdot (k-1)! \cdot (\Delta-1)^{(k+d)/2} \cdot n + c \cdot (n+m) \\
&\overset{(**)}{\leq} C \cdot k! \cdot (\Delta-1)^{(k+d)/2} \cdot n
\end{aligned}$$

Inequality $(*)$ holds, since $(\Delta-1)+(k-1) \leq (\Delta-1)\cdot(k-1)$ for all $k \geq 3$ and $\Delta \geq 3$. Inequality $(**)$ holds, since $2c \leq C$, $k \geq 3$, and $m \leq (\Delta-1) \cdot n$.

Suppose that there is no vertex $v \in S$ with $N(v) \subseteq S$. By the induction hypothesis and due to **(3)**,

$$\begin{aligned}
T(k, k+d) \leq \quad & \frac{k^2}{2} \cdot (\Delta-1) \cdot T(k-2, k+d-2) + c \cdot 2^k \cdot (\Delta-1)^{(k+d)/2} \cdot k^3 \cdot n \\
\leq \quad & C \cdot \frac{k^2}{2} (\Delta-1) \cdot (k-2)! \cdot (\Delta-1)^{(k+d)/2-1} \cdot n \\
& + \; c \cdot 2^k \cdot (\Delta-1)^{(k+d)/2} \cdot k^3 \cdot n \\
\leq \quad & C \cdot \left( \frac{k^2}{2} \cdot (k-2)! + \frac{1}{128} \cdot 2^k \cdot k^3 \right) \cdot (\Delta-1)^{(k+d)/2} \cdot n \\
\overset{(*)}{\leq} \quad & C \cdot k! \cdot (\Delta-1)^{(k+d)/2} \cdot n
\end{aligned}$$

Inequality $(*)$ holds, since $C = 128c$ and $\frac{k^2}{2} \cdot (k-2)! + \frac{1}{128} \cdot 2^k \cdot k^3 \leq k!$ for all $k \geq 3$.

Hence, the whole algorithm runs in $\mathcal{O}(k! \cdot (\Delta-1)^{(k+d)/2} \cdot n)$ time. □

Note that the above running time is (besides the change from the $2^k$ factor to a $k!$ factor in the running time) a direct generalization of the previous best algorithm for LS VC which runs in $\mathcal{O}(2^k \cdot (\Delta-1)^{k/2} \cdot k \cdot n)$ time [98] to GLS VC.

### 3.3.2 Parameterizing Weighted Gap Local Search by Maximum Degree

Next, we consider Gap LS Weighted Vertex Cover when parameterized by the maximum degree $\Delta(G)$ plus $k$.

**Proposition 3.25.** *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of* GLS WVC. *One can enumerate all valid connected $k$-swaps for $S$ in $G$ in $\mathcal{O}(2^k \cdot (\Delta - 1)^k \cdot k^3 \cdot n)$ time.*

*Proof sketch.* We adapt an algorithm of Katzmann and Komusiewicz [98]. Note that the algorithm by Katzmann and Komusiewicz [98] does not consider the weights of vertices and only enumerates $k$-swaps $W$ with $|W \cap S| = |W \setminus S| + 1$. Unfortunately, in weighted graphs, a valid improving swap $W$ does not have to fulfill $|W \cap S| = |W \setminus S| + 1$. Hence, we cannot directly apply the algorithm of Katzmann and Komusiewicz [98] for weighted graphs.

The idea of our algorithm is to enumerate, for each vertex $v \in S$ the valid connected $k$-swaps $W$ containing $v$. In the following, call the vertices in the vertex cover $S$ *black* and the vertices of the independent set $V \setminus S$ *white*. Recall that if a black vertex $u$ is contained in a swap $W$, then no black neighbors of $u$ may be included in $W$ and all white neighbors of $u$ must be included in $W$. In other words, white vertices are added automatically, when they are neighbors of the current swap. Now, to extend a current swap $W$, we pick a white vertex $u$ in $W$ and choose 1) to exclude all neighbors of $u$ outside of $W$ to be swapped in all recursive calls or 2) to pick one of the at most $\Delta - 1$ neighbors of $u$ outside of $W$ to be added to $W$. In Case 1), we "finish" a white vertex, in the at most $\Delta - 1$ subcases of Case 2), we add a black vertex. The total number of white and black vertices is at most $k$, and thus we may abort the search at a search tree depth of $k$. The total search tree size is thus $\mathcal{O}(\Delta^k) = \mathcal{O}(2^k \cdot (\Delta - 1)^k)$ for each of the $\mathcal{O}(n)$ initial vertices $v$. We omit the discussion of the further polynomial parts of the running time. □

Since an instance $I$ of LS WVC has a good swap if and only if it has a *connected* good swap (see Observation 3.5), Proposition 3.25 implies that LS WVC can be solved in the following running time.

**Corollary 3.26.** LS WVC *can be solved in $\mathcal{O}(2^k \cdot (\Delta - 1)^k \cdot k^3 \cdot n)$ time.*

To solve the gap version GLS WVC we again encounter the problem that the sought good swap is not necessarily connected. Hence, for GLS WVC we show a related algorithm to the one we presented for GLS VC using only one branching

rule. This rule is, more or less, an adaptation of Branching Rule 3.3.2 to the weighted version. Consider a good swap $W$ for $I$. This time, we want to find some valid improving $j$-swap $W_j$ for $S$ in $G$ for each $j \in [1, k]$ and branch into the cases of either swapping $W_j$ or swapping some neighbor of $W_j$.

Unfortunately, a result similar to Proposition 3.21 cannot be obtained, that is, in the weighted case, each good swap might contain only one neighbor of $W_1$.[1] Hence, in the worst case, each of these branching cases reduces the parameter $k$ only by one which would lead to a running time factor of $(\Delta - 1)^{2k}$ instead of $(\Delta - 1)^k$. Our goal is, thus, to decrease the number of cases in which the parameter is only reduced by one. To this end, we analyze the swap $W_1$ separately. Let $S_1 := \{v \in S \mid N(v) \subseteq S\}$ denote the set of vertices of improving 1-swaps for $S$ in $G$ and let $v^*$ be the unique vertex of $W_1$. Since $v^*$ is some vertex in $S_1$ of highest weight, if $W \cap N[v^*] = W \cap N[W_1] = \emptyset$, then we can replace some distinct vertex $w^*$ of $S_1$ contained in $W$ by $v^*$ and also obtain a good swap for $I$. Hence, we can then reduce our branching cases for $j \geq 2$ to the ones in which we consider either swapping $W_j$ or some neighbor of $W_j$ which is not contained in $S_1$. Since the remaining considered swaps for $j \geq 2$ have size at least 2, only $|N[W_1]| \leq \Delta + 1$ cases remain in which the parameter is only decreased by one.

For each $i \in [1, k]$, let $W_j$ denote some valid connected swap of size exactly $j$ for $I$ with maximal improvement, if such a swap exists. We call a collection $\{W_j\}_1^k$ of such swaps a *weighted swap family*. Recall that $S_1 := \{v \in S \mid N(v) \subseteq S\}$ denotes the set of vertices of improving 1-swaps for $S$ in $G$.

**Proposition 3.27.** *Let $I = (G = (V, E), \omega, S, k, d)$ be a yes-instance of* GLS WVC *and let $\{W_j\}_1^k$ be a weighted swap family. There is a good swap $W$ for $I$ such that* (i) *$W$ is connected or* (ii) *$W \cap N[W_1] \neq \emptyset$ or* (iii) *there is some $j \in [2, \lfloor \frac{k}{2} \rfloor]$ such that $W_j \subseteq W$ or $(W \cap N(W_j)) \setminus S_1 \neq \emptyset$.*

*Proof.* Since $I$ is a yes-instance of GLS WVC, there is a minimum good swap $W$ for $I$. Suppose that $W$ is not connected, $W \cap N[W_1] = \emptyset$, for each $j \in [2, \lfloor \frac{k}{2} \rfloor]$ where $W_j$ exists, $W_j \not\subseteq W$ and $W \cap N(W_j) \setminus S_1 = \emptyset$, as otherwise the statement already holds. Let $C$ be the smallest connected component in $G[W]$ and let $\ell := |C|$. Since $W$ is not connected, $\ell \leq \lfloor \frac{k}{2} \rfloor$. Note that $W' := W \setminus C$ is a valid $(d - \delta(C))$-improving $(k - \ell)$-swap for $S$ in $G$. Since $C$ is a connected swap of size exactly $\ell$, the swap $W_\ell$ exists. Recall that $W_\ell$ is some valid connected swap of size exactly $\ell$ for $I$ that maximizes $\delta(W_\ell)$. Hence, the improvement $\delta(W_\ell)$ of $W_\ell$ is at least the

[1] Consider the path $(u, v, w)$, with $\omega(v) = 3$, and $\omega(u) = \omega(w) = 1$. Let $S = \{u, v\}$, $k = d = 2$. The only 2-improving 2-swap is $\{v, w\}$. Note that this swap avoids the only valid improving 1-swap $\{u\}$ and contains only one neighbor of $u$.

improvement $\delta(C)$ of $C$. In the following, we show that $W^* := W' \cup W_\ell$ is a good swap for $I$.

Recall that $W_\ell \not\subseteq W$ and $(N(W_\ell) \cap W) \setminus S_1 = \emptyset$. Further, since each vertex of $S_1 \cap W$ is isolated in $G[W]$, $W_\ell \cap W$ is a proper subset of $W_\ell$ and $W_\ell$ is a valid swap for $S$ in $G$. Hence, $W_\ell \cap W = \emptyset$, as otherwise $C$ is not the smallest connected component in $G[W]$ by the fact that $W \cap N(W_\ell) \setminus S_1 = \emptyset$. Note that $W^*$ is a $d$-improving $k$-swap for $S$ in $G$. It remains to show that $W^*$ is valid. Since $W'$ and $W_\ell$ are both valid, it follows that $W^*$ is valid if $W' \cap N(W_\ell \cap S) = \emptyset$. By assumption, this is the case for $\ell = 1$. Moreover by assumption, $W' \cap N(W_\ell \cap S) \setminus S_1 = \emptyset$ if $\ell > 1$. Note that since every valid connected swap containing some vertex of $S_1$ has size 1, $W' \cap S_1 \neq \emptyset$ would then contradict the assumption that $C$ is the smallest connected swap in $G[W]$ with $|C| = \ell > 1$. □

Hence, we derive the following branching rule.

**Branching Rule 3.3.3.** Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS WVC such that there is no connected good swap for $I$. Moreover, let $\{W_j\}_1^k$ be a given weighted swap family. If $W_1$ exists, branch into the case of swapping $W$ for each swap $W \in \{\{w\} \mid w \in N(W_1)\}$. Additionally, for each $j \in [1, \lfloor \frac{k}{2} \rfloor]$ where $W_j$ exists, branch into the case of swapping $W$ for each swap $W \in \{W_j\} \cup \{\{w\} \mid w \in N(W_j) \setminus S_1\}$.

With this branching rule, we are now able to show the algorithm for GLS WVC.

**Theorem 3.28.** GLS WVC *can be solved in* $\mathcal{O}(k! \cdot (\Delta - 1)^k \cdot n)$ *time.*

*Proof.* Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS VC. If $k \leq 2$, then we can solve $I$ in $\mathcal{O}(n+m)$ time due to Lemma 3.9. If $\Delta(G) = 2$, then the treewidth of $G$ is at most 2 and we can compute a nice tree decomposition of $G$ of width at most 2 in $\mathcal{O}(n+m)$ time and afterwards solve $I$ in $\mathcal{O}((2^k+k^2) \cdot n+m) \subseteq \mathcal{O}(k! \cdot n+m)$ time due to Theorem 3.15. Hence, we can assume in the following, that $k \geq 3$ and $\Delta(G) \geq 3$.

The algorithm now works as follows. First, compute a weighted swap family $\{W_j\}_1^k$. Due to Proposition 3.25, this can be done in $\mathcal{O}(2^k \cdot (\Delta - 1)^k \cdot k^3 \cdot n)$ time. Now, if there is some $j \in [1, k]$ such that $W_j$ exists and $\delta(W_k) \geq d$, then $I$ is a yes-instance of GLS WVC. Otherwise, there is no connected good swap $W$ for $I$. Compute the set $S_1 := \{v \in S \mid N(v) \subseteq S\}$ of possible improving swaps of size 1 and apply Branching Rule 3.3.3. Due to Proposition 3.27, this is correct.

It remains to show the running time. Let $\mathcal{I}_1$ denote the set of swap-instances considered during one application of Branching Rule 3.3.3 where a vertex of $N[W_1]$

is swapped and let $\mathcal{I}_{>1}$ denote the remaining swap-instances considered during one application of Branching Rule 3.3.3. That is,

$$\mathcal{I}_1 := \begin{cases} \{\text{swap}(I, \{w\}) \mid w \in N[W_1]\} & W_1 \text{ exists}, \\ \emptyset & \text{otherwise}, \end{cases}$$

and

$$\mathcal{I}_{>1} := \bigcup_{\substack{j \in [2, \lfloor \frac{k}{2} \rfloor] \\ W_j \text{ exists}}} \{\text{swap}(I, W_j)\} \cup \{\text{swap}(I, \{w\}) \mid w \in N(W_j) \setminus S_1\}.$$

Note that $\mathcal{I}_1$ has size at most $\Delta + 1$ and $k' \leq k - 1$ for each instance $I' \in \mathcal{I}_1$. Next, we bound the size of $\mathcal{I}_{>1}$. Since for each $j \in [2, \lfloor \frac{k}{2} \rfloor]$ for which $W_j$ exists, $W_j$ has size $j$ and each vertex $v \in W_j$ has at most $\Delta - 1$ neighbors outside of $W_j$, we can upper-bound the size of $|\mathcal{I}_{>1}|$ as

$$|\mathcal{I}_{>1}| \leq \sum_{j=2}^{\lfloor \frac{k}{2} \rfloor} (j \cdot (\Delta - 1) + 1) \leq \frac{\frac{k}{2} \cdot (\frac{k}{2} + 1)}{2} \cdot (\Delta - 1) + \frac{k}{2} - \Delta = \frac{k^2 + 2k}{8} \cdot (\Delta - 1) + \frac{k}{2} - \Delta.$$

Note that for each instance of $\mathcal{I}_{>1}$, we have $k' \leq k - 2$. This is due to the fact that for each $j \geq 2$ where $W_j$ exists, $|W_j| \geq 2$ and $N(w) \setminus S \neq \emptyset$ for each vertex $w \in N(W_j) \setminus S_1$.

Hence, the recurrence for the running time is

$$T(k) \leq \left( \frac{k^2 + 2k}{8} \cdot (\Delta - 1) + \frac{k}{2} - \Delta \right) \cdot T(k - 2) + (\Delta + 1) \cdot T(k - 1) + \mathcal{O}(2^k \cdot (\Delta - 1)^k \cdot k^2 \cdot n).$$

We show by induction over $k$ that there is a constant $C$ such that $T(k) \leq C \cdot k! \cdot (\Delta - 1)^k \cdot n$.

First, observe that there is a constant $c$ such that

**(1)** GLS WVC can be solved in $c \cdot (n + m)$ time if $k < 3$,

**(2)** $T(k) \leq (\frac{k^2+2k}{8} \cdot (\Delta - 1) + \frac{k}{2} - \Delta) \cdot T(k-2) + (\Delta + 1) \cdot T(k-1) + c \cdot 2^k \cdot (\Delta - 1)^k \cdot k^3 \cdot n$.

We set $C := 700c$. Hence, for the base case of $k < 3$ the statement holds directly due to **(1)**. As the inductive step, we show that the statement holds for $k$ if it holds for $k - j$ for each $j \in [1, k - 1]$.

By the induction hypothesis and due to **(2)**,

$$\begin{aligned}
T(k) &\leq \left(\frac{k^2+2k}{8}\cdot(\Delta-1)+\frac{k}{2}-\Delta\right)\cdot T(k-2)\\
&\quad + (\Delta+1)\cdot T(k-1) + c\cdot 2^k\cdot(\Delta-1)^k\cdot k^3\cdot n\\
&\leq C\cdot\left(\frac{k^2+2k}{8}\cdot(\Delta-1)+\frac{k}{2}-\Delta\right)\cdot(k-2)!\cdot(\Delta-1)^{k-2}\cdot n\\
&\quad + C\cdot(\Delta+1)\cdot(k-1)!\cdot(\Delta-1)^{k-1}\cdot n + c\cdot 2^k\cdot(\Delta-1)^k\cdot k^3\cdot n\\
&= C\cdot(k-2)!\cdot(\Delta-1)^{k-2}\\
&\quad\cdot\left(\frac{k^2+2k}{8}\cdot(\Delta-1)+\frac{k}{2}-\Delta+(k-1)\cdot(\Delta^2-1)\right)\cdot n\\
&\quad + c\cdot 2^k\cdot(\Delta-1)^k\cdot k^3\cdot n\\
&\overset{(*)}{\leq} C\cdot(k-2)!\cdot(\Delta-1)^k\cdot(k\cdot(k-1)-1)\cdot n + c\cdot 2^k\cdot(\Delta-1)^k\cdot k^3\cdot n\\
&\leq C\cdot k!\cdot(\Delta-1)^k\cdot n - C\cdot(k-2)!\cdot(\Delta-1)^k\cdot n + c\cdot 2^k\cdot(\Delta-1)^k\cdot k^3\cdot n\\
&\overset{(**)}{\leq} C\cdot k!\cdot(\Delta-1)^k\cdot n
\end{aligned}$$

Inequality $(*)$ holds, since $\frac{k^2+2k}{8}\cdot(\Delta-1)+\frac{k}{2}-\Delta+(k-1)\cdot(\Delta^2-1) \leq (\Delta-1)^2\cdot(k\cdot(k-1)-1)$ for all $k\geq 3$ and $\Delta\geq 3$. Inequality $(**)$ holds, since $C=700c$ and $(k-2)!\geq\frac{1}{700}\cdot 2^k\cdot k^3$.

Hence, the whole algorithm runs in $\mathcal{O}(k!\cdot(\Delta-1)^k\cdot n)$ time. □

### 3.3.3 Parameterizing the Weighted Gap Version by *h*-Index

Finally, we show that we can replace $\Delta(G)$ in the above running time by the $h$-index of $G$. Recall that the *$h$-index* $h(G)$ of a graph $G$ is the largest integer such that $G$ contains at least $h(G)$ vertices of degree at least $h(G)$. The idea behind this algorithm is to branch on all possibilities on how a potential improving swap may intersect the set of high-degree vertices. For each of these potential intersections $X$, we compute the corresponding swap-instance and solve it with the help of Theorem 3.28 after removing the remaining high-degree vertices. Intuitively, we want to avoid all possible valid swaps that contain any of the high-degree vertices outside of $X$. For each high-degree vertex $v$ outside of $X$, we have to consider two cases: If $v$ is contained in $S$, we can simply remove $v$. Otherwise, if $v\notin S$, we additionally have to remove each neighbor of $v$. This is due to the fact that each valid swap $W$ that swaps any neighbor of $v$ out of the vertex cover $S$ has to also swap $v$ into the vertex cover.

Based on this observation, we define for each vertex set $V' \subseteq V$ an *exclusion instance $I'$ for $V'$ and $I$* as the instance of GLS WVC, where all vertices of $V' \cup N(V' \setminus S)$ are removed from $I$. Formally, let $I = (G, \omega, S, k, d)$ be an instance of GLS WVC, then the exclusion instance $I'$ of GLS WVC for $V'$ and $I$ is defined as $I' := (G', \omega, S', k, d)$, where $G' := G - (V' \cup N(V' \setminus S))$ and $S' := S \cap V(G')$. By the above argumentation, we derive the following property for exclusion instances.

**Lemma 3.29.** *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of* GLS WVC *and let $V' \subseteq V$ be a set of vertices. There is a good swap $W$ for $I$ with $W \cap V' = \emptyset$ if and only if the exclusion instance of $V'$ and $I$ is a yes-instance of* GLS WVC*.*

*Proof.* Let $I' = (G', S', k, d)$ be the exclusion instance of $V'$ and $I$. Let $W$ be a good swap for $I$ that avoids $V'$. Since $W$ is valid, $W$ contains no vertex of $N(V' \setminus S)$. Hence, $W$ is a good swap for $I'$.

Let $W$ be a good swap for $I'$. By definition of the exclusion instance $I'$, $W$ is also a $d$-improving $k$-swap for $S$ in $G$ which avoids $V'$. It remains to show that $S \oplus W$ is a vertex cover of $G$. This is the case if no vertex of $V' \setminus S$ is adjacent to some vertex in $W$. By construction of $I'$, there is no vertex in $V(G')$ which is adjacent to some vertex in $V' \setminus S$. Hence $W$ is a good swap for $I$ that avoids $V'$. □

Based on the concept of exclusion instances and due to Theorem 3.28, we are now able to show the algorithm for GLS WVC when parameterized by $h(G)$ plus $k$.

**Theorem 3.30.** GLS WVC *can be solved in $\mathcal{O}(k! \cdot (h-1)^k \cdot n)$ time.*

*Proof.* Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS WVC. If $k \leq 2$, the statement holds due to Lemma 3.9. Otherwise, we compute the $h$-index $h(G)$ of $G$ and the set of vertices $H$ with degree at least $h(G) + 1$ in $\mathcal{O}(n + m)$ time. If $h(G) = 2$, then the treewidth of $G$ is at most 4 and we can compute a nice tree decomposition of $G$ of width at most 4 in $\mathcal{O}(n + m)$ time and afterwards solve $I$ in $\mathcal{O}((4^k + k^2) \cdot n + m) \subseteq \mathcal{O}(k! \cdot n + m)$ time due to Theorem 3.15. Hence, we assume in the following that $k \geq 3$ and $h(G) \geq 3$. Since $h(G)$ is the $h$-index of $G$, $H$ contains at most $h(G)$ vertices. The idea is to consider all possibilities for the intersection of a good swap $W$ with $H$ and afterwards solve the resulting swap-instance with the algorithm of Theorem 3.28. To obtain a linear running time for instances where the $h$-index and $k$ are both constants, we initially compute the adjacency matrix of $G[H]$ in $\mathcal{O}(h(G)^2 + m)$ time. For each $W_H \subseteq H$ of size at most $k$, we check whether $W_H \cap S$ is an independent set and, if this is the case, compute the swap-instance $I' := (G', \omega', S', k', d') := \text{swap}(I, W_H)$ in $\mathcal{O}(k^2)$ time. Note that if $W_H$ has size exactly $k$, then $k' \leq 0$ and, thus, $I'$ is a trivial instance of GLS WVC and can

be solved in $\mathcal{O}(1)$ time. If for such a swap $W_H$ of size $k$, $\text{swap}(I, W_H)$ is a yes-instance of GLS WVC, answer yes. Otherwise, suppose that $W_H$ has size less than $k$. Now, do the following. Since we search for a good swap $W$ for $I$ with $W \cap H = W_H$, no other vertex of $H$ is contained in $W$. Hence, we can ignore choices of $W_H$, where $N(W_H \cap S) \cap H \not\subseteq W_H$. For each remaining choice of $W_H$, we can compute the exclusion instance $I_{W_H}$ of $H \cap V(G')$ and $I'$ in $\mathcal{O}(h(G) \cdot n)$ time. Recall that due to Lemma 3.29, $I_{W_H}$ is a yes-instance of GLS WVC if and only if there is a good swap for $S'$ that avoids $H \cap V(G')$. Let $G_{W_H}$ denote the graph of $I_{W_H}$. Since $G_{W_H}$ is a subgraph of $G$ and contains only vertices of $V \setminus H$, $G_{W_H}$ has a maximum degree of $h(G)$. Moreover, note that $k' \leq k - |W_H|$. Consequently, $I_{W_H}$ can be solved in $\mathcal{O}((k - |W_H|)! \cdot (h(G) - 1)^{k-|W_H|} \cdot n)$ time due to Theorem 3.28. Note that this dominates the running time to compute $I_{W_H}$ since $|W_H| < k$. Afterwards, we answer yes if and only if there is some considered swap $W_H \subseteq H$ of size at most $k - 1$ such that $I_{W_H}$ is a yes-instance of GLS WVC.

Hence, the running time of the described algorithm can be upper bounded by

$$\mathcal{O}(n + m + h(G)^2) + \mathcal{O}((h(G) - 1)^k \cdot k \cdot n) + \sum_{j=1}^{\min(k-1, h(G))} \mathcal{O}(h(G) - 1)^j / j^2) \cdot (\mathcal{O}(k^2 + h(G) \cdot n) + \mathcal{O}((k - j)! \cdot (h(G) - 1)^{k-j} \cdot n) + m)) \text{ time},$$

since due to Lemma 3.14, for each $j \geq 1$, $H$ contains at most $\mathcal{O}((h(G) - 1)^j / j^2)$ subsets of size at most $j$. Since $m \in \mathcal{O}(h(G) \cdot n)$ and $\sum_{j=1}^{k} \frac{1}{j^2} \in \mathcal{O}(1)$, we obtain the stated running time. □

## 3.4 Using Modular Decompositions

Next, we provide FPT-algorithms that use modular decompositions which, roughly speaking, provide a hierarchical view of the different neighborhoods in a graph $G$.

### 3.4.1 Modular Decompositions

A *modular decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{T}, \beta)$ consisting of a rooted tree $\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*)$ with root $x^* \in \mathcal{V}$ and a function $\beta$ that maps each node $x \in \mathcal{V}$ to a graph $\beta(x)$ [124]. If $x$ is a leaf of $\mathcal{T}$, then $\beta(x)$ contains a single vertex of $V$ and for each vertex $v \in V$, there is exactly one leaf $\ell$ of $\mathcal{T}$ such that the graph $\beta(\ell)$ consists only of $v$. If $x$ is not a leaf node, then the vertex set of $\beta(x)$ is exactly the set of child nodes of $x$ in $\mathcal{T}$. Moreover, let $V_x$ denote the set of vertices of $V$ contained

in leaf nodes of the subtree rooted in $x$. Formally, $V_x$ is defined as $V(\beta(\ell))$ for leaf nodes $\ell$ and recursively defined as $\bigcup_{y \in V(\beta(x))} V_y$ for each non-leaf node $x$. Moreover, we define $G_x = (V_x, E_x) := G[V_x]$. A modular decomposition has the property that for each non-leaf node $x$ and any pair of distinct nodes $y \in V(\beta(x))$ and $z \in V(\beta(x))$, $y$ and $z$ are adjacent in $\beta(x)$ if and only if there is an edge in $G$ between each pair of vertices of $V_y$ and $V_z$ and $y$ and $z$ are not adjacent if and only if there is no edge in $G$ between any pair of vertices of $V_y$ and $V_z$. Hence, it is impossible that there are vertex pairs $(v_1, w_1) \in V_y \times V_z$ and $(v_2, w_2) \in V_y \times V_z$ such that $v_1$ is adjacent to $w_1$ and $v_2$ is not adjacent to $w_2$.

We call $\beta(x)$ the *quotient graph* of $x$. A quotient graph is *prime* if there is no set $A \subseteq V(\beta(x))$ with $2 \leq |A| < |V(\beta(x))|$ such that all vertices of $A$ have the same neighborhood in $V(\beta(x)) \setminus A$. The *width of a modular decomposition* is the size of a largest vertex set of any quotient graph and the *modular-width* of a graph $G$, denoted by $\mathrm{mw}(G)$, is the minimal width of any modular decomposition of $G$.

### 3.4.2 Parameterization by Modular-Width

We now provide a dynamic programming algorithm over the modular decomposition of $G$. The nodes of the decomposition are processed in a bottom-up manner. The idea is to consider for a node $x$ the possibilities of how a swap may interact with the vertex sets that are represented by the vertices $y$ of $\beta(x)$. We use the fact that any valid swap of $G$ must also correspond in the natural way to a valid swap of $\beta(x)$. More precisely, if some vertex in the set represented by $y$ goes to the independent set, then the vertex cover must include the vertex set represented by $z$ for all neighbors $z$ of $y$ in $\beta(x)$.

**Theorem 3.31.** GLS WVC *can be solved in* $\mathcal{O}(\mathrm{mw}(G)^k \cdot k \cdot (\mathrm{mw}(G)+k) \cdot n + m)$ *time.*

*Proof.* Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS WVC. First, we compute a modular decomposition $(\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*), \beta)$ of minimal width in $\mathcal{O}(n + m)$ time [124]. Note that $\mathcal{T}$ has $\mathcal{O}(n)$ nodes. Next, we describe a dynamic program on the modular decomposition $(\mathcal{T}, \beta)$ to solve GLS WVC.

For each node $x \in \mathcal{V}$ in the modular decomposition, we have a dynamic programming table $D_x$. The table $D_x$ has entries of type $D_x[k']$ for $k' \in [0, k]$. Each entry $D_x[k']$ stores the maximal improvement $\delta_S(W)$ of a valid $k'$-swap $W \subseteq V_x$ for $S \cap V_x$ in $G_x$.

Next, we describe how to fill the dynamic programming tables. Let $\ell$ be a leaf node of $\mathcal{T}$ and let $v$ be the unique vertex of $V(\beta(\ell)) = V_\ell$. We fill the table $D_\ell$ by

setting

$$D_\ell[k'] := \begin{cases} 0 & v \notin S \vee k' = 0 \\ \omega(v) & v \in S \wedge k' > 0 \end{cases}$$

for each $k' \in [0, k]$.

To compute the entries for all remaining nodes $x$ of $\mathcal{T}$, we use an auxiliary table $Q_{S_x}$. Let $S_x$ be an independent set in $\beta(x)$ and let $S_x(i)$ denote the $i$th vertex of $S_x$ according to some arbitrary but fixed ordering with $i \in [1, |S_x|]$. Moreover, let $V_x^{\leq i} = \bigcup_{j=1}^{i} V_{S_x(j)}$. Recall that $S_x(j)$ is both a vertex of $\beta(x)$ and a child node of $x$ in $\mathcal{T}$ and that $V_{S_x(j)}$ denotes the set of all vertices of $G$ that are contained in the subtree of $\mathcal{T}$ rooted in $S_x(j)$. The dynamic programming table $Q_{S_x}[i, k']$ has entries for $i \in [1, |S_x|]$ and $k' \in [0, k]$ and stores the maximal improvement of a valid $k'$-swap $W$ for $S \cap V_x^{\leq i}$ in $G[V_x^{\leq i}]$, such that for each $j \in [1, i]$, at least one vertex of $S \cap V_{S_x(j)}$ is contained in $W$. We set

$$Q_{S_x}[i, k'] := \begin{cases} -\infty & k' < i, \\ D_{S_x(1)}[k'] & i = 1, \text{and} \\ \max_{1 \leq k'' \leq k'} D_{S_x(i)}[k''] + Q_{S_x}[i-1, k'-k''] & \text{otherwise.} \end{cases}$$

Since we are looking for a $k'$-swap $W$ that contains for each $j \in [1, i]$ at least one vertex of $S \cap V_{S_x(j)}$, the value of the table is set to $-\infty$ if $k' < i$, since there is no such swap of size at most $k' < i$. If $k' \geq i$, then the value of the table is determined by finding the best way to swap $k''$ vertices of $V_{S_x(i)}$ and $k' - k''$ vertices of $V_x^{\leq i-1}$.

The entries for $D_x$ can then be computed as follows:

$$D_x[k'] := \max_{\substack{S_x \subseteq V(\beta(x)) \\ |W^*| \leq k' \\ S_x \text{ is independent}}} Q_{S_x}[1, k' - |W^*|] - \omega(W^*)$$

where $W^* := \bigcup_{y \in N_x(S_x)} (V_y \setminus S)$. Since we are looking for a swap $W$ where for each $y \in S_x$, at least one vertex of $S \cap V_y$ is contained in $W$ and thus leaves the vertex cover, each vertex of $W^*$ has to be added to obtain a vertex cover.

The maximal improvement of any valid $k$-swap for $S$ in $G$ can be found in $D_{x^*}[k]$, where $x^*$ is the root of the modular decomposition. Moreover, the corresponding $k$-swap can be found via traceback.

Next, we analyze the running time. For each non-leaf node $x$, and each independent set $S_x$ of size at most $k$ in $\beta(x)$, there are $\mathcal{O}(k^2)$ table entries in $Q_{S_x}$ and each of these entries can be computed in $\mathcal{O}(k)$ time. Recall that for a set of size $x$, $\binom{x}{\leq k}$ denote the number of different subsets of size at most $k$.

Since each quotient graph has $\mathcal{O}(\binom{\mathrm{mw}(G)}{\leq k})$ many independent sets of size at most $k$, all entries of all tables $Q_{S_x}$ can be computed in $\mathcal{O}(\binom{\mathrm{mw}(G)}{\leq k} \cdot k^3 \cdot n)$ time, since the modular decomposition has $\mathcal{O}(n)$ quotient graphs. For each node $x$, there are $\mathcal{O}(k)$ table entries in $D_x$. We will show that we can compute each of them in $\mathcal{O}(k^2 \cdot (\mathrm{mw}(G) + k))$ time. To this end, we precompute for each node $x$ the size $|V_x \setminus S|$ and the weight $\omega(V_x \setminus S)$ to compute $|W^*|$ and $\omega(W^*)$ in $\mathcal{O}(k)$ time afterwards. Since for all non-leaf nodes $x$, $V_x \setminus S = \bigcup_{y \in V(\beta(x))}(V_y \setminus S)$, we can compute $|V_x \setminus S|$ as $\sum_{y \in V(\beta(x))} |V_y \setminus S|$ and $\omega(V_x \setminus S)$ as $\sum_{y \in V(\beta(x))} \omega(V_y \setminus S)$. This can be done in $\mathcal{O}(\mathrm{mw}(G) \cdot n)$ time since the modular decomposition has $\mathcal{O}(n)$ quotient graphs. Hence, for an independent set $S_x$ of size at most $k$, we can compute $|W^*|$ and $\omega(W^*)$ in $\mathcal{O}(k)$ time. Since we can enumerate all subsets $S_x$ of size at most $k$ of $V(\beta(x))$ in $\mathcal{O}(\binom{\mathrm{mw}(G)}{\leq k})$ time and check in $\mathcal{O}(\mathrm{mw}(G) \cdot k)$ time if $S_x$ is independent in $\beta(x)$, we can compute $D_x[k']$ in $\mathcal{O}(\binom{\mathrm{mw}(G)}{\leq k} \cdot k^2 \cdot (k + \mathrm{mw}(G)))$ time. Consequently, we can compute all entries of the dynamic programming tables in $\mathcal{O}(\binom{\mathrm{mw}(G)}{\leq k} \cdot k^3 \cdot (k + \mathrm{mw}(G)) \cdot n + m)$ time, which is $\mathcal{O}(\mathrm{mw}(G)^k \cdot k \cdot (\mathrm{mw}(G) + k) \cdot n + m)$ time due to Lemma 3.14.

Since the value of $Q_{S_x}[1, k']$ is only evaluated once during the whole computation of this dynamic programming algorithm, we can remove the table $Q_{S_x}$ after evaluating $Q_{S_x}[1, k']$ for each $k'$. Consequently, this algorithm also only uses polynomial space. □

Note that with a slight modification of the dynamic programming algorithm, we can improve the running time for the special case of GLS VC. Recall that $I$ is a yes-instance of GLS VC if and only if, there is a good swap $W$ for $I$ with $|W \cap S| \leq \frac{k+d}{2}$. Hence for GLS VC, it is sufficient in the computation of $D_x[k']$ to only check for independent sets $S_x$ in $\beta(x)$ of size at most $\min(k', \frac{k+d}{2})$. Thus, we obtain the following.

**Corollary 3.32.** GLS VC *can be solved in* $\mathcal{O}(\mathrm{mw}(G)^{\frac{k+d}{2}} \cdot k \cdot (\mathrm{mw}(G)+k) \cdot n + m)$ *time.*

## 3.5 Concluding Remarks

In this chapter, we analyzed the parameterized complexity for LS Vertex Cover and the more general versions Gap LS Vertex Cover, LS Weighted Vertex Cover, and Gap LS Weighted Vertex Cover. We showed that these problems can be solved in $\ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time for $\ell$ being (i) the treewidth, (ii) the maximum degree, (iii) the $h$-index, and (iv) the modular-width of the input graph. We complemented the running time upper-bounds by showing that for all considered

parameter $\ell$ a running time of $\ell^{\Omega(k)} \cdot n^{\mathcal{O}(1)}$ time is necessary, under the assumption that the ETH holds. These upper and lower bounds apply for both the strict and the permissive version of the problems.

**Open questions.** There are numerous possibilities for future research. First, it seems interesting to study further parameterized local search problems with the aim of achieving FPT-algorithms whose running times grow strongly only with respect to the operational parameter $k$. For example, we assume that some of the techniques presented in this chapter can be generalized to local search versions of Hitting Set and 0-1 Integer Linear Programming. This question could also be relevant in other scenarios with operational parameters, for example in turbo-charging of greedy algorithms [2, 39, 68]. Second, it is open to improve the running time for the considered problems since our conditional lower bounds are not completely tight. For example, for LS Vertex Cover parameterized by $k$ and the $h$-index, it is open whether a running time of $\mathcal{O}(h^{k/2} \cdot n)$ is possible. Third, it would be interesting to explore gap versions of further local search problems, both from a theoretical and a practical perspective. In this context it would be interesting to explore whether there are parameters $\ell$ for which the normal parameterized local search problem admits an FPT-algorithm but the gap version is W[1]-hard. Furthermore, it would be interesting to identify structural parameters $\ell$ where permissive local search has an FPT-algorithm with running time $\ell^{g(k)} \cdot n^{\mathcal{O}(1)}$ and strict local search does not.

Finally, it is open to further explore the concrete practical potential of our results for Vertex Cover: Can our theoretical results lead to good implementations of parameterized local search for Weighted Vertex Cover? An implementation of the algorithm behind Corollary 3.26 yielded very good results as a post-processing for state-of-the-art algorithms [164]. Still, it would be interesting to evaluate whether for example the algorithm parameterized by the $h$-index and $k$ presented in this chapter perform similarly well for Weighted Vertex Cover. Moreover, can the performance of the successful parameterized local search algorithm for unweighted Vertex Cover with parameter $(\Delta, k)$ [98] be improved by some of the techniques presented in this chapter?

# Chapter 4

# Parameterized Local Search for Max $c$-Cut

Graph coloring and its generalizations are among the most famous NP-hard optimization problems [97] with numerous practical applications [94]. In one prominent problem variant, we want to color the vertices of an edge-weighted graph with $c$ colors so that the sum of the weights of all edges that have endpoints with different colors is maximized. This problem is known as Max $c$-Cut [62, 96] or Maximum Colorable Subgraph [140]. Applications of Max $c$-Cut include data clustering [31, 55], computation of rankings [31], design of experimental studies [9], sampling of public opinions in social networks [91], channel assignment in wireless networks [159], and module detection in genetic interaction data [114]. In addition, Max $c$-Cut is closely related to the energy minimization problem in Hopfield neural networks [102, 169, 170]. An equivalent formulation of the problem is to ask for a coloring that minimizes the weight sum of the edges whose endpoints receive the same color; this formulation is known as Generalized Graph Coloring [168]. The main difference is that for instances of Generalized Graph Coloring, one usually assumes that all edge weights are non-negative, whereas for Max $c$-Cut, one usually also allows negative weights.

From an algorithmic viewpoint, even restricted cases of Max $c$-Cut are hard: The special case $c = 2$ is the Max Cut problem which is NP-hard already for unit weights [65, 97], even on graphs with maximum degree 3 [17]. Moreover, for all $c \geq 3$ the Graph Coloring problem, where we ask for a coloring of the vertices with $c$ colors such that the endpoints of every edge receive different colors, is NP-hard [97]. As a consequence, Max $c$-Cut is NP-hard for all $c \geq 3$, again even when all edges have unit weight and where we want each edges to have endpoints of different

color. While Max $c$-Cut admits polynomial-time constant factor approximation algorithms [62], it does not admit polynomial-time approximation schemes unless P = NP [140], even on graphs with bounded maximum degree [17].

Due to these hardness results, Max $c$-Cut is mostly solved using heuristic approaches [57, 114, 120, 178]. A popular heuristic for Max $c$-Cut is hill-climbing local search with respect to the $k$-flip neighborhood for $k \in \{1, 2\}$ [57, 114]. The 1-flip neighborhood has received interest from a theoretical standpoint: For Max Cut finding a 1-(flip)-optimal solution is PLS-complete on edge-weighted graphs [151] and thus presumably not efficiently solvable in the worst case. This PLS-completeness result for the 1-flip neighborhood was later extended to Generalized Graph Coloring, and thus to Max $c$-Cut, for all $c \geq 2$ [168]. For graphs where the absolute values of all edge weights are constant, however, a simple hill-climbing algorithm terminates after $\mathcal{O}(m)$ improvements, where $m$ is the number of edges in the input graph. This is due to the fact that each improvement made by the hill-climbing algorithm increases the objective value of the solution by at least 1 and the maximum possible objective value of any solution is $\mathcal{O}(m)$.

Again, to avoid being stuck in bad local optima, we aim to find efficient algorithms to search the $k$-flip neighborhood for larger values of $k$. As noted by Kleinberg and Tardos [102], a standard algorithm for searching the $k$-flip neighborhood takes $\Theta(n^k \cdot m)$ time where $n$ is the number of vertices. This led Kleinberg and Tardos to conclude that the $k$-flip neighborhood is impractical. In this chapter, we ask whether we can do better than the brute-force $\Theta(n^k \cdot m)$-time algorithm or, in other words, whether the dismissal of $k$-flip neighborhood may have been premature. So far, the parameterized complexity of searching the $k$-flip neighborhood for Max $c$-Cut was only considered by Fellows at al. [52]. They showed that on graphs of bounded local treewidth, one can search the $k$-flip neighborhood for Max $c$-Cut in FPT-time. Their algorithm implies that one can search the $k$-flip neighborhood for Max $c$-Cut in $2^{\Delta^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$ time.

**Our results.** We study LS Max $c$-Cut, where we want to decide whether a given coloring has a better one in its $k$-flip neighborhood. First, we show that LS Max $c$-Cut is presumably not solvable in $f(k) \cdot n^{o(k)}$ time for any computable function $f$.

Afterwards, we present an algorithm with running time $\mathcal{O}((3e\Delta)^k \cdot c \cdot k^3 \cdot \Delta \cdot n)$, where $\Delta$ is the maximum degree of the input graph. The algorithm is based on two main observations: First, we show that minimal improving flips are connected in the input graph. This allows to enumerate candidate flips in $\mathcal{O}((e\Delta)^k \cdot k \cdot n)$ time due to Lemma 2.16. Second, we show that, given a set of $k$ vertices to flip, we can deter-

mine an optimal way to flip their colors in $\mathcal{O}(3^k \cdot c \cdot k^2 + k \cdot \Delta)$ time. We then discuss several ways to speed up the algorithm, for example by computing upper bounds for the improvement of partial flips. We finally evaluate our algorithm experimentally when it is applied as post-processing for a state-of-the-art Max $c$-Cut heuristic [120]. In this application, we take the solutions computed by the heuristic and improve them by hill-climbing with the $k$-flip neighborhood for increasing values of $k$. We show that, for a standard benchmark data set, a large fraction of the previously best solutions can be improved by our algorithm, leading to new record solutions for these instances. The post-processing is particularly successful for instances with $c > 2$ and both positive and negative edge weights.

**Formal problem definition.** Let $G = (V, E)$ be an undirected graph. In this chapter, we may refer to a coloring of $V$ as a *coloring of* $G$. Let $\chi$ be a $c$-coloring of $G$, we define the set $E(\chi)$ of *properly colored edges* as $E(\chi) := \{\{u, v\} \in E \mid \chi(u) \neq \chi(v)\}$.

Recall that for an edge-weight function $\omega\colon E \to \mathbb{Q}$ and an edge set $E' \subseteq E$, we let $\omega(E')$ denote the total weight of all edges in $E'$. LS Max $c$-Cut is now formally defined as follows.

> LS Max $c$-Cut
> **Input:** A graph $G = (V, E)$, $c \in \mathbb{N}$, a weight function $\omega\colon E \to \mathbb{Q}$, a $c$-coloring $\chi$, and $k \in \mathbb{N}$.
> **Question:** Is there a $c$-coloring $\chi'$ with $d_{\text{flip}}(\chi, \chi') \leq k$ and $\omega(E(\chi')) > \omega(E(\chi))$?

The special case of LS Max $c$-Cut where $c = 2$ can alternatively be defined as follows by using partitions instead of colorings.

> LS Max Cut
> **Input:** A graph $G = (V, E)$, a weight function $\omega\colon E \to \mathbb{Q}$, a partition $(A, B)$ of $V$, and $k \in \mathbb{N}$.
> **Question:** Is there a set $S \subseteq V$ of size at most $k$ such that $\omega(E(A, B)) < \omega(E(A \oplus S, B \oplus S))$?

While these problems are defined as decision problems, our algorithms solve the search problem that returns an improving $k$-flip if it exists.

Let $\chi$ and $\chi'$ be $c$-colorings of $G$. If $\omega(E(\chi)) > \omega(E(\chi'))$, we say that $\chi$ is *improving* over $\chi'$ (with respect to LS Max $c$-Cut). In this chapter, we call a $c$-coloring $\chi$ *$k$-(flip-)optimal* if $\chi$ has no improving $k$-neighbor $\chi'$. We say that $\chi'$ is

an *inclusion-minimal improving k-flip for* $\chi$, if $\chi'$ is an improving $k$-neighbor of $\chi$ and if there is no improving $k$-neighbor $\widetilde{\chi}$ of $\chi$ with $D_{\text{flip}}(\chi, \widetilde{\chi}) \subsetneq D_{\text{flip}}(\chi, \chi')$. Let $(A, B)$ be a partition of $G$. In the context of LS Max Cut, we call a vertex set $S$ *inclusion-minimal improving k-flip for* $(A, B)$, if $|S| \leq k$, $\omega(E(A \oplus S, B \oplus S)) > \omega(E(A, B))$, and if there is no vertex set $S' \subsetneq S$ such that $\omega(E(A \oplus S', B \oplus S')) > \omega(E(A, B))$.

## 4.1 W[1]-hardness and a Tight ETH Lower Bound for LS Max *c*-Cut and Related Problems

We first show our intractability result for LS Max Cut. More precisely, we show that LS Max Cut is W[1]-hard when parameterized by $k$ even on bipartite graphs with unit weights. This implies that even on instances where an optimal partition can be found in linear time, LS Max Cut presumably cannot be solved within $f(k) \cdot n^{\mathcal{O}(1)}$ time for any computable function $f$. Afterwards, we extend the intractability results even to the permissive version of LS Max $c$-Cut on general graphs. Finally, we can then also derive new intractability results for local search versions for the related partition problems Min Bisection, Max Bisection, and Max Sat.

To prove the intractability results for the strict version, we introduce the term of *blocked vertices* in instances with unit weights. Intuitively, a vertex $v$ is blocked for a color class $i$ if we can conclude that $v$ does not move to $i$ in any optimal $k$-neighbor of the current solution just by considering the graph neighborhood of $v$. This concept is formalized as follows.

**Definition 4.1.** Let $G = (V, E)$ be a graph, let $\chi$ be a $c$-coloring of $G$, and let $k$ be an integer. Moreover, let $v$ be a vertex of $V$ and let $i \in [1, c] \setminus \{\chi(v)\}$ be a color. The vertex $v$ is *$(i, k)$-blocked in $G$ with respect to $\chi$* if $v$ has at least $2k+1$ more neighbors of color $i$ than of color $\chi(v)$ with respect to $\chi$, that is, if $|\{w \in N(v) \mid \chi(w) = i\}| \geq |\{w \in N(v) \mid \chi(w) = \chi(v)\}| + 2k - 1$.

Note that a partition $P := (B_1, B_2)$ can be interpreted as the 2-coloring $\chi_P$ defined for each vertex $v \in V$ by $\chi_P(v) := i$, where $i$ is the unique index of $\{1, 2\}$ such that $v \in B_i$. Hence, we may also say that a vertex $v$ is *$(B_i, k)$-blocked in $G$ with respect to $(B_1, B_2)$*, if $v$ is $(i, k)$-blocked in $G$ with respect to $\chi_P$.

### 4.1.1 Hardness for the Strict Version of LS Max *c*-Cut

**Lemma 4.2.** *Let $G = (V, E)$ be a graph, let $\chi$ be a $c$-coloring of $G$, let $k$ be an integer. Moreover, let $v$ be a vertex in $V$ which is $(i, k)$-blocked in $G$ with respect to $\chi$. Then, there is no inclusion-minimal improving $k$-neighbor $\chi'$ of $\chi$ with $\chi'(v) = i$.*

*Proof.* Let $\chi'$ be a $c$-coloring of $G$ with $d_{\text{flip}}(\chi, \chi') \leq k$ and $\chi'(v) = i$. Hence, $v \in D_{\text{flip}}(\chi, \chi')$ and thus $D_{\text{flip}}(\chi, \chi')$ contains at most $k-1$ neighbors of $v$. Consequently, at most $k-1$ more neighbors of $v$ receive color $\chi(v)$ under $\chi'$ than under $\chi$. Similarly, at most $k-1$ more neighbors of $v$ receive color $i$ under $\chi$ than under $\chi'$. Since $v$ is $(i,k)$-blocked in $G$ with respect to $\chi$, this then implies that $v$ has more neighbors of color $i$ than of color $\chi(v)$ under $\chi'$. Let $\chi^*$ be the $c$-coloring of $G$ that agrees with $\chi'$ on all vertices of $V \setminus \{v\}$ and where $\chi^*(v) := \chi(v)$. Note that $E(\chi') \setminus E(\chi^*) = \{\{w,v\} \in E \mid \chi'(w) = \chi(v)\}$ and $E(\chi^*) \setminus E(\chi') = \{\{w,v\} \in E \mid \chi'(w) = i\}$. This implies that $\chi^*$ is a better $c$-coloring for $G$ than $\chi'$, since

$$|E(\chi^*)| - |E(\chi')| = |E(\chi^*) \setminus E(\chi')| - |E(\chi') \setminus E(\chi^*)| > 0.$$

Hence, $\chi'$ is not an inclusion-minimal improving $k$-neighbor of $\chi$, since $D_{\text{flip}}(\chi, \chi^*) = D_{\text{flip}}(\chi, \chi') \setminus \{v\} \subsetneq D_{\text{flip}}(\chi, \chi')$. □

The idea of blocking a vertex by its neighbors finds application in the construction for the W[1]-hardness from the next theorem.

**Theorem 4.3.** LS Max Cut *is* W[1]*-hard when parameterized by* $k$ *on bipartite* 2*-degenerate graphs with unit weights.*

*Proof.* We reduce from Clique. Recall that Clique is W[1]-hard when parameterized by the size $k$ of the sought clique [35, 44].

Let $I := (G = (V, E), k)$ be an instance of Clique. We construct an equivalent instance $I' := (G' = (V', E'), \omega', A', B', k')$ of LS Max Cut with $\omega' : E' \to \{1\}$ as follows. We start with an empty graph $G'$ and add each vertex of $V$ to $G'$. Next, for each edge $e \in E$, we add two vertices $u_e$ and $w_e$ to $G'$ and make both $u_e$ and $w_e$ adjacent to each endpoint of $e$ in $G'$. Afterwards, we add a vertex $v^*$ to $G'$ and for each edge $e \in E$, we add vertices $x_e$ and $y_e$ and edges $\{w_e, x_e\}$, $\{w_e, y_e\}$, and $\{x_e, v^*\}$ to $G'$. Finally, we add a set $V_z$ of $|E| - 2 \cdot \binom{k}{2} + 1$ vertices to $G'$ and make each vertex of $V_z$ adjacent to $v^*$.

In the following, for each $\alpha \in \{u, w, x, y\}$, let $V_\alpha$ denote the set of all $\alpha$-vertices in $G'$, that is, $V_\alpha := \{\alpha_e \mid e \in E\}$. We set

$$B' := V_w \cup \{v^*\} \cup V_z,\ A' := V' \setminus B', \text{ and}$$
$$k' := 2 \cdot \binom{k}{2} + k + 1.$$

To ensure that some vertices are blocked in the final graph $G'$, we add the following further vertices to $A'$ and $B'$: For each vertex $v' \in V_u \cup V_y$, we add a set

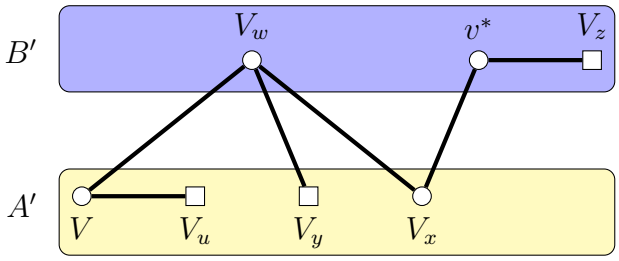



**Figure 4.1:** The connections between the different vertex sets in $G'$. Two vertex sets $X$ and $Y$ are adjacent in the figure if $E(X, Y) \neq \emptyset$. Each vertex $v$ in a vertex set with a rectangular node is $k'$-blocked from the opposite part of the partition. The vertex set $V_\Gamma$ is not shown.

of $2k' + 2$ vertices to $B'$ that are only adjacent to $v'$ and for each vertex $v' \in V_z$, we add a set of $2k' + 2$ vertices to $A'$ that are only adjacent to $v'$. Let $V_\Gamma$ be the set of those additional vertices. Figure 4.1 shows a sketch of the vertex sets and their connections in $G'$. Note that $G'$ is bipartite and 2-degenerate.

Note that each vertex in $V_u \cup V_y$ is contained in $A'$, has at most two neighbors in $A'$, and at least $2k' + 2$ neighbors in $B'$. Moreover, each vertex in $V_z$ is contained in $B'$, has one neighbor in $B'$, and $2k' + 2$ neighbors in $A'$. Hence, each vertex in $V_u \cup V_y$ is $(B', k')$-blocked and each vertex in $V_z$ is $(A', k')$-blocked. Consequently, due to Lemma 4.2, no inclusion-minimal improving $k'$-flip for $(A', B')$ contains any vertex of $V_u \cup V_y \cup V_z$. As a consequence, no inclusion-minimal improving $k'$-flip for $(A', B')$ contains any vertex of $V_\Gamma$. In other words, only vertices in $V$, $V_w$, $V_x$, and the vertex $v^*$ can flip their colors.

Before we show the correctness, we provide some intuition. By the above, intuitively, a clique $S$ in the graph $G$ then corresponds to a flip of vertex $v^*$, the vertices of $S$, and the vertices $w_e$ and $x_e$ for each edge $e$ of the clique. The key mechanism is that each inclusion-minimal improving flip has to contain $v^*$, so that edges between $v^*$ and $V_z$ become properly colored. To compensate for the edges between $V_x$ and $v^*$ that are not properly colored after flipping $v^*$, for some edges $e$ of $G$, the corresponding vertices of $V_x$ and $V_w$ and both endpoints of $e$ have to flip their color. The size of $V_z$ ensures that this has to be done for at least $\binom{k}{2}$ such edges of $G$. Since we only allow a flip of size $k'$, this then ensures that the edges of $G$ whose corresponding vertices flip their color belong to a clique of size $k$ in $G$.

Next, we show that $I$ is a yes-instance of Clique if and only if $I'$ is a yes-instance of LS Max Cut.

($\Rightarrow$) Let $S \subseteq V$ be a clique of size $k$ in $G$. Hence, $\binom{S}{2} \subseteq E$. We set $S' := S \cup$

$\{w_e, x_e \mid e \in \binom{S}{2}\} \cup \{v^*\}$. Note that $S'$ has size $k + 2 \cdot \binom{k}{2} + 1 = k'$. Let $C := E(A', B')$ and let $C' := E(A' \oplus S', B' \oplus S')$. It remains to show that $C'$ contains more edges than $C$. To this end, note that $C$ and $C'$ differ only on edges that have at least one endpoint in $S'$.

First, we discuss the edges incident with at least one vertex of $S = S' \cap V$. For each vertex $v \in S$ and each vertex $v' \in N_G(v) \setminus S$, the edge $\{v, w_{\{v,v'\}}\}$ is contained in $C$ but not in $C'$ and the edge $\{v, u_{\{v,v'\}}\}$ is contained in $C'$ but not in $C$. For each other neighbor $v' \in N_G(v) \cap S$, the edge $\{v, u_{\{v,v'\}}\}$ is contained in $C'$ but not in $C$ and the edge $\{v, w_{\{v,v'\}}\}$ is contained in both $C$ and $C'$. Next, we discuss the remaining edges incident with some vertex of $\{w_e, x_e \mid e \in \binom{S}{2}\}$. For each edge $e \in \binom{S}{2} \subseteq E$, the edges $\{w_e, x_e\}$ and $\{x_e, v^*\}$ are contained in both $C$ and $C'$ and the edge $\{w_e, y_e\}$ is contained in $C$ but not in $C'$. Finally, we discuss the remaining edges incident with $v^*$. For each edge $e \in E \setminus \binom{S}{2}$, the edge $\{v^*, x_e\}$ is contained in $C$ but not in $C'$ and for each vertex $z \in V_z$, the edge $\{v^*, z\}$ is contained in $C'$ but not in $C$. Hence,

$$\begin{aligned} C \setminus C' = \; & \{\{v, w_{\{v,v'\}}\} \mid v \in S, v' \in N_G(v) \setminus S\} \\ & \cup \{\{w_e, y_e\} \mid e \in \binom{S}{2}\} \\ & \cup \{\{v^*, x_e\} \mid e \in E \setminus \binom{S}{2}\}. \end{aligned}$$

Furthermore, we have

$$C' \setminus C = \{\{v, u_{\{v,v'\}}\} \mid v \in S, v' \in N_G(v)\} \cup \{\{v^*, z\} \mid z \in V_z\}.$$

Since $|V_z| = |E| - 2 \cdot \binom{k}{2} + 1$, we get

$$\begin{aligned} |C' \setminus C| - |C \setminus C'| = \; & |\{\{v, u_{\{v,v'\}}\} \mid v \in S, v' \in N_G(v)\}| \\ & - |\{\{v, w_{\{v,v'\}}\} \mid v \in S, v' \in N_G(v) \setminus S\}| \\ & + |\{\{v^*, z\} \mid z \in V_z\}| \\ & - |\{\{w_e, y_e\} \mid e \in \binom{S}{2}\}| - |\{\{v^*, x_e\} \mid e \in E \setminus \binom{S}{2}\}| \\ = \; & 2 \cdot \binom{k}{2} + |E| - 2 \cdot \binom{k}{2} + 1 - |E| = 1. \end{aligned}$$

Consequently, $C'$ contains exactly one edge more than $C$. Hence, $I'$ is a yes-instance of LS Max Cut.

($\Leftarrow$) Let $S' \subseteq V'$ be an inclusion-minimal improving $k'$-flip for $(A', B')$. Due to Lemma 4.2, we can assume that $S' \subseteq V \cup V_w \cup V_x \cup \{v^*\}$ since all other vertices

of $V' \setminus V_\Gamma$ are blocked from the opposite part of the partition and for each vertex $x \in V_\Gamma$, the unique neighbor of $x$ in $G'$ is thus not contained in $S'$. By construction of $G'$, each vertex $v \in V$ is adjacent to $|N_G(v)|$ vertices of $A'$ and adjacent to $|N_G(v)|$ vertices of $B'$. Since $S'$ is inclusion-minimal and contains no vertex of $\{u_e \mid e \in E\}$, for each vertex $v \in S' \cap V$, there is at least one edge $e \in E$ incident with $v$ in $G$ such that $S'$ contains the vertex $w_e$, as otherwise, removing $v$ from $S'$ still results in an even better partition than $(A' \oplus S', B' \oplus S')$, that is,

$$|E(A' \oplus (S' \setminus \{v\}), B' \oplus (S' \setminus \{v\}))| \geq |E(A' \oplus S', B' \oplus S')| > |E(A', B')|.$$

Moreover, recall that $B'$ contains all vertices of $V_w$ and each vertex $w_e \in V_w$ is adjacent to the four vertices $\{x_e, y_e\} \cup e$ of $A'$ and is adjacent to no vertex of $B'$. Since $S'$ is inclusion-minimal and contains no vertex of $V_y$, for each vertex $w_e \in S' \cap V_w$, all three vertices of $\{x_e\} \cup e$ are contained in $S'$, as otherwise, removing $w_e$ from $S'$ does not result in a worse partition than $(A' \oplus S', B' \oplus S')$, that is,

$$|E(A' \oplus (S' \setminus \{w_e\}), B' \oplus (S' \setminus \{w_e\}))| \geq |E(A' \oplus S', B' \oplus S')| > |E(A', B')|.$$

Furthermore, $A'$ contains all vertices of $V_x$ and each vertex $x_e \in V_x$ is adjacent to the vertices $w_e$ and $v^*$ in $B'$ and adjacent to no vertex in $A'$. Since $S'$ is inclusion-minimal, for each vertex $x_e \in S' \cap V_x$, both $w_e$ and $v^*$ are contained in $S'$, as otherwise, removing $x_e$ from $S'$ does not result in a worse partition than $(A' \oplus S', B' \oplus S')$, that is,

$$|E(A' \oplus (S' \setminus \{x_e\}), B' \oplus (S' \setminus \{x_e\}))| \geq |E(A' \oplus S', B' \oplus S')| > |E(A', B')|.$$

Note that the above statements imply that $S'$ contains $v^*$. This is due to the facts that

a) $S'$ is non-empty,

b) $S'$ contains only vertices of $V \cup V_w \cup V_x \cup \{v^*\}$,

c) if $S'$ contains a vertex of $V$, then $S'$ contains a vertex of $V_w$,

d) if $S'$ contains a vertex of $V_w$, then $S'$ contains a vertex of $V_x$, and

e) if $S'$ contains a vertex of $V_x$, then $S'$ contains the vertex $v^*$.

Recall that $v^*$ is adjacent to the $|E|$ vertices $V_x$ in $A'$ and to the $|E| - 2 \cdot \binom{k}{2} + 1$ vertices $V_z$ in $B'$. Hence, since $S'$ is inclusion-minimal and no vertex of $V_z$ is contained in $S'$, $S'$ contains at least $\binom{k}{2}$ vertices of $V_x$, as otherwise,

$$|E(A' \oplus (S' \setminus \{v^*\}), B' \oplus (S' \setminus \{v^*\}))| \geq |E(A' \oplus S', B' \oplus S')| > |E(A', B')|.$$

Concluding, $S'$ contains $v^*$ and for at least $\binom{k}{2}$ edges $e \in E$ the vertices $x_e$, $w_e$, and the endpoints of $e$. Let $S := S' \cap V$. Since $S'$ has size at most $k' = 2 \cdot \binom{k}{2} + k + 1$, the above implies that $S$ has size at most $k$. Since $S'$ contains the endpoints of at least $\binom{k}{2}$ edges $e \in E$, we conclude that $S$ is a clique of size $k$ in $G$. Hence, $I$ is a yes-instance of Clique. □

This implies that even on instances where an optimal solution can be found in polynomial time, local optimality cannot be verified efficiently. This property was was also shown for LS VC. Namely, LS VC was shown to be W[1]-hard with respect to the search radius even on bipartite graphs [69].

Next, we describe how to adapt the above reduction can to prove W[1]-hardness of LS Max $c$-Cut for each fixed $c \geq 2$ when parameterized by $k$.

Consider the instance $I := (G, \omega, (A, B), k)$ of LS Max Cut that has been constructed in the proof of Theorem 4.3 and let $c > 2$. For every vertex $v$ of $G$, we add further degree-one neighbors. More precisely, for every color $i \in [3, c]$, the vertex $v$ receives additional neighbors of color $i$ such that $v$ is $(i, k)$-blocked. Let $G'$ be the resulting graph.

Then, for any inclusion-minimal improving $k$-flip $\chi'$ for $\chi$ of $G'$, we have $S := D_{\text{flip}}(\chi, \chi') \subseteq A \cup B$, $\chi'(a) = 2$ for each $a \in A \cap S$, and $\chi'(b) = 1$ for each $b \in B \cap S$. Hence, $I$ is a yes-instance of LS Max Cut if and only if the instance $I'$ is a yes-instance of LS Max $c$-Cut. Since we only added degree-one vertices, the graph is still bipartite and 2-degenerate.

**Corollary 4.4.** *For every $c \geq 2$,* LS Max $c$-Cut *is* W[1]*-hard when parameterized by $k$ on bipartite 2-degenerate graphs with unit weights.*

### 4.1.2 Hardness for the Permissive Version of LS Max $c$-Cut

Next, we present a running time lower bound for LS Max $c$-Cut based on the ETH. This lower bound holds even for the permissive version of LS Max $c$-Cut.

**Lemma 4.5.** *Even the permissive version of* LS Max Cut *does not admit an* FPT*-algorithm when parameterized by $k$, unless* FPT = W[1] *and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. More precisely, this hardness holds even if there is an optimal solution in the $k$-flip neighborhood of the initial solution.*

*Proof.* We reduce from Independent Set. Independent Set is W[1]-hard when parameterized by the size $k$ of the sought independent set even if the size of a largest independent set in the input graph is at most $k$ [35, 44]. Furthermore, even under

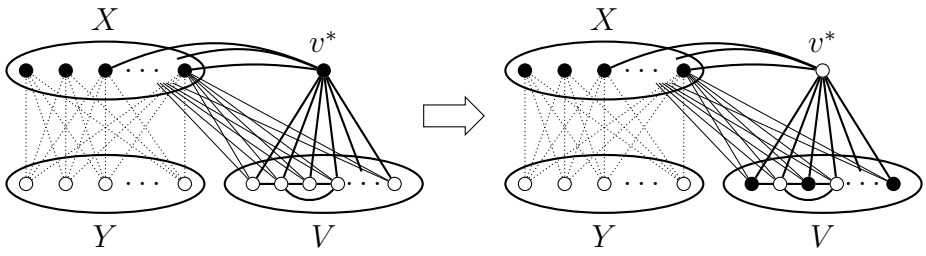



**Figure 4.2:** Two solution for the instance of LS Max Cut constructed in the proof of Lemma 4.5. In both solutions, the parts of the respective partitions are indicated by the color of the vertices. The left partition shows the initial solution and the right partition shows an improving solution, if one exists. The flip between these partitions is an independent set of size $k$ in $G$ together with the vertex $v^*$.

these restrictions, Independent Set cannot be solves in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails [35].

Let $I := (G = (V, E), k)$ be an instance of Independent Set and let $n := |V|$ and $m := |E|$. We construct an equivalent instance $I' := (G' = (V', E'), \omega', A, B, k')$ of LS Max Cut with $\omega' : E' \to \{1\}$ as follows. We initialize $G'$ as a copy of $G$. Next, we add two vertex sets $X$ and $Y$ of size $n^3$ each to $G'$. Additionally, we add a vertex $v^*$ to $G'$. Next, we describe the edges incident with at least one newly added vertex. We add edges to $G'$ such that $v^*$ is adjacent to each vertex of $V$ and $n-k+1$ arbitrary vertices of $X$. Moreover, we add edges to $G'$ such that each vertex of $X$ is adjacent to each vertex of $Y$. Finally, we add edges to $G'$ such that each vertex $v \in V$ is adjacent with exactly $|N_G(v)|$ arbitrary vertices of $X$ in $G'$. This completes the construction of $G'$. It remains to define the initial partition $(A, B)$ of $V'$ and the search radius $k'$. We set $A := V \cup Y$, $B := X \cup \{v^*\}$, and $k' := k+1$. This completes the construction of $I'$. Note that each vertex of $V$ has exactly one neighbor more in $B$ than in $A$. Moreover, $v^*$ has $2k-1$ more neighbors in $A$ than in $B$.

Intuitively, the only way to improve over the partition $(A, B)$ is to flip an independent set in $G$ of size $k$ from $A$ to $B$. Then, $v^*$ has exactly one neighbor more in $B$ than in $A$ and flipping $v^*$ from $B$ to $A$ improves over the initial partition $(A, B)$ by exactly one edge. See Figure 4.2 for an illustration. Next, we show that this reduction is correct.

($\Rightarrow$) Let $S$ be an independent set of size $k$ in $G$. We set $A' := (A \setminus S) \cup \{v^*\}$ and $B' := V' \setminus A' = (B \cup S) \setminus \{v^*\}$ and show that $(A', B')$ improves over $(A, B)$.

Note that $E(A,B) = E(Y,X) \cup E(V,\{v^*\}) \cup E(V,X)$ which implies that

$$|E(A,B)| = |Y| \cdot |X| + |V| + \sum_{v \in V} |N_G(v)| = n^6 + n + 2m.$$

Moreover, note that

$$E(A',B') = E(Y,X) \cup E(\{v^*\},S) \cup E(\{v^*\},X) \cup E(V \setminus S,S) \cup E(V \setminus S,X).$$

Since $S$ is an independent set in $G$ and $G'$, for each vertex $v \in S$, $V \setminus S$ contains all neighbors of $v$ in $G$. Consequently, $|E(V \setminus S,S)| = \sum_{v \in S} |N_G(v)|$. Since $S$ has size $k$, the above implies that

$$\begin{aligned} |E(A',B')| &= |Y| \cdot |X| + |S| + n - k + 1 + \sum_{v \in S} |N_G(v)| + \sum_{v \in V \setminus S} |N_G(v)| \\ &= n^6 + n + 2m + 1. \end{aligned}$$

Hence, the partition $(A',B')$ improves over the partition $(A,B)$ which implies that $I'$ is a yes-instance of LS Max Cut.

($\Leftarrow$) Let $(A',B')$ be an optimal partition of $G'$ and suppose that $(A',B')$ improves over $(A,B)$. We show that $I$ is a yes-instance of Independent Set. Due to the first direction, this then implies that $I'$ is a yes-instance of LS Max Cut which further implies that the initial partition $(A,B)$ is an optimal partition for $G'$ if and only if $(A,B)$ is $k'$-flip optimal. To show that $I$ is a yes-instance of Independent Set, we first analyze the structure of the optimal partition $(A',B')$ for $G'$.

First, we show that all vertices of $X$ are on the opposite part of the partition $(A',B')$ than all vertices of $Y$.

**Claim 1.** $A'$ contains all vertices of $Y$ and $B'$ contains all vertices of $X$, or $A'$ contains all vertices of $X$ and $B'$ contains all vertices of $Y$.

*Proof of Claim.* We show that if neither of these statements holds, then $E(A',B')$ contains less edges than $E(A,B)$. This would then contradict the fact that $(A',B')$ is an optimal partition. We distinguish two cases.

If all vertices of $X \cup Y$ are on the same part of the partition $(A',B')$, then $E(A',B')$ contains at most $|N_{G'}(V)| + |N_{G'}(v^*)| < n^3$ edges. Hence, $E(A',B')$ contains strictly less edges than $E(A,B)$ which contradicts the fact that $(A',B')$ is an optimal partition. Otherwise, if not all vertices of $X \cup Y$ are on the same part of the partition $(A',B')$ and not all vertices of $X$ are on the opposite part of the partition than all vertices of $Y$, then both $A'$ and $B'$ contain at least one vertex of $X$ each, or both $A'$ and $B'$ contain at least one vertex of $Y$ each. In both cases, at least $\min(|X|,|Y|) =$

$n^3$ edges of $E(X, Y)$ are not contained in $E(A', B')$. Hence, $E(A', B')$ has size at most $|E| - n^3$. Again, since $E$ contains at most $|N_{G'}(V)| + |N_{G'}(v^*)| < n^3$ edges outside of $E(X, Y)$, this implies that $E(A', B')$ contains strictly less edges than $E(A, B)$. This contradicts the fact that $(A', B')$ is an optimal partition. Consequently, the statement holds. ■

By Claim 1, we may assume without loss of generality that $A'$ contains all vertices of $Y$ and $B'$ contains all vertices of $X$. Next, we show that $v^*$ is contained in $A'$. Assume towards a contradiction that $v^*$ is contained in $B'$. Hence, each vertex $v \in V$ has $|N_G(v)| + 1$ neighbors in $B'$. By construction, each vertex $v \in V$ has exactly $2 \cdot |N_G(v)| + 1$ neighbors in $G'$. Hence, each vertex of $V$ has more neighbors in $B'$ than in $A'$. Consequently, $A'$ contains all vertices of $V$, since $(A', B')$ is an optimal partition for $G'$. This implies that $A' := Y \cup V = A$ and $B' := X \cup \{v^*\} = B$ which contradicts the assumption that $(A', B')$ improves over $(A, B)$. Consequently, $v^*$ is contained in $A'$ together with all vertices of $Y$. It remains to determine the partition of the vertices of $V$ into $A'$ and $B'$.

Let $S := B' \cap V$. We show that $S$ is an independent set of size $k$ in $G$. First, assume towards a contradiction that $S$ is not an independent set in $G$. Then, there are two adjacent vertices $u$ and $w$ of $V$ in $B'$. Hence, $u$ has at least $|N_G(u)| + 1$ neighbors in $B'$, since $u$ is adjacent to $|N_G(u)|$ vertices of $X$. Since the degree of $u$ in $G'$ is $2 \cdot |N_G(u)| + 1$, flipping vertex $u$ from $B'$ to $A'$ would result in an improving solution. This contradicts the fact that $(A', B')$ is an optimal partition of $G'$. Hence, $S$ is an independent set in $G$. By assumption, the size of the largest independent set in $G$ is at most $k$. Hence, to show that $S$ is an independent set of size $k$, it suffices to show that $S$ has size at least $k$. To this end, we analyze the number of edges of $E(A', B')$. Recall that $A' := (A \setminus S) \cup \{v^*\}$ and $B' := (B \cup S) \setminus \{v^*\}$. Hence, analogously to the first direction of the correctness proof, $E(A', B') = E(Y, X) \cup E(\{v^*\}, S) \cup E(\{v^*\}, X) \cup E(V \setminus S, S) \cup E(V \setminus S, X)$. Since $S$ is an independent set, this implies that

$$\begin{aligned} |E(A', B')| &= |Y| \cdot |X| + |S| + n - k + 1 + \sum_{v \in S} |N_G(v)| + \sum_{v \in V \setminus S} |N_G(v)| \\ &= n^6 + n - k + 1 + 2m + |S|. \end{aligned}$$

Since we assumed that the partition $(A', B')$ improves over $(A, B)$ and $|E(A, B)| = n^6 + n + 2m$, this implies that $S$ has size at least $k$. Consequently, $S$ is an independent set of size $k$ in $G$, which implies that $I$ is a yes-instance of Independent Set.

This also implies that, if $(A, B)$ is not an optimal partition for $G'$, then there is an optimal partition for $G'$ with flip-distance exactly $k'$ from $(A, B)$. Hence, the re-

duction is correct. Recall that Independent Set is W[1]-hard when parameterized by $k$ and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. Since $|V'| \in \mathcal{O}(n^3)$ and $k' \in \mathcal{O}(k)$, this implies that the permissive version of LS Max Cut (i) does not admit an FPT-algorithm when parameterized by $k'$, unless FPT = W[1] and (ii) cannot be solved in $f(k') \cdot |V'|^{o(k')}$ time for any computable function $f$, unless the ETH fails. □

These intractability results can be transferred to each larger constant value of $c$.

**Theorem 4.6.** *For every $c \geq 2$, even the permissive version of* LS Max $c$-Cut *does not admit an* FPT*-algorithm when parameterized by $k$, unless* FPT = W[1] *and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. This holds even on graphs with unit weights.*

*Proof.* Let $I := (G = (V, E), \omega, A, B, k)$ be an instance of LS Max Cut with $\omega(e) = 1$ for each edge $e \in E$, such that there is an optimal partition $(A', B')$ for $G$ with flip-distance at most $k$ with $(A, B)$ and let $n := |V|$ and $m := |E|$. Due to Lemma 4.5, even under these restrictions, LS Max Cut does not admit an FPT-algorithm when parameterized by $k$, unless FPT = W[1] and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails.

Fix a constant $c > 2$. We describe how to obtain in polynomial time an equivalent instance $I' := (G' := (V', E'), c, \omega', \chi, k)$ of LS Max $c$-Cut. We obtain the graph $G'$ by adding for each $i \in [1, c]$ an independent set $X_i$ of size $n^2$ to $G$ and adding edges such that each vertex of $X_i$ is adjacent with each vertex of $\{v \in X_j \mid j \in [1, c] \setminus \{i\}\}$. Additionally, for each $i \in [3, c]$, we add edges between each vertex of $X_i$ and each vertex of $V$. This completes the construction of $G'$. Again, each edge receives weight 1 with respect to the weight function $\omega'$. Finally, we define the $c$-coloring $\chi$ of $G'$. For each $i \in [1, c]$, we set $\chi(v) := i$ for each vertex $v \in X_i$. Additionally, for each vertex $v \in A$, we set $\chi(v) := 1$ and for each vertex $w \in B$, we set $\chi(w) := 2$. This completes the construction of $I'$.

Note that $E'(\chi)$ contains all edges of $E' \setminus E$ and thus misses less than $n^2$ edges of $G'$ in total. Intuitively, this ensures that only vertices of $V$ may flip their color and only to the colors 1 or 2, since in each other $c$-coloring, at least $n^2$ edges are missing. In other words, the large independent sets $X_i$ ensure that to improve over $\chi$, one can only flip vertices of $V$ from color 1 to 2 or vice versa. This is then improving if and only if the corresponding flip on the LS Max Cut-instance $I$ is improving.

Next, we show the correctness of the reduction.

($\Rightarrow$) Let $(A', B')$ be an optimal partition of $G$ that improves over $(A, B)$ and let $S$ be the flip between $(A, B)$ and $(A', B')$. By assumption, we know that $S$ has size at

most $k$. We define a $c$-coloring $\chi^*$ for $G'$ as follows: The colorings $\chi$ and $\chi^*$ agree on all vertices of $V' \setminus S$, for each vertex $v \in A \cap S$, we set $\chi^*(v) := 2$, and for each vertex $w \in B \cap S$, we set $\chi^*(w) := 1$. Note that $D_{\text{flip}}(\chi, \chi^*) = S$ which implies that $\chi^*$ and $\chi$ have flip-distance at most $k$. Moreover, note that $A' = \{v \in V \mid \chi^*(v) = 1\}$ and $B' = \{v \in V \mid \chi^*(v) = 2\}$. It remains to show that $\chi^*$ improves over $\chi$. To this end, note that both $E'(\chi)$ and $E'(\chi^*)$ contain all edges of $E' \setminus E$. Hence, $\chi^*$ improves over $\chi$ if and only if $|E'(\chi^*) \cap E| > |E'(\chi) \cap E|$. Note that $E'(\chi^*) \cap E = E(A', B')$ and $E'(\chi) \cap E = E(A, B)$. Hence, the assumption that $(A', B')$ is a better partition for $G$ than $(A, B)$ implies that $\chi^*$ improves over $\chi$. Consequently, $I'$ is a yes-instance of LS Max $c$-Cut.

($\Leftarrow$) Let $\chi^*$ be an optimal $c$-coloring for $G'$ and assume that $\chi^*$ improves over $\chi$. To show that there is a better partition for $G$ than $(A, B)$, we first prove that each optimal $c$-coloring $\chi^*$ for $G'$ contains all edges of $E' \setminus E$.

**Claim 2.** It holds that $E'(\chi^*)$ contains all edges of $E' \setminus E$.

*Proof of Claim.* Assume towards a contradiction that $E'(\chi^*)$ does not contain all edges of $E' \setminus E$. Since $E'(\chi^*)$ does not contain all edges of $E' \setminus E$, there is an $i \in [1, c]$ and a vertex $x \in X_i$, such that at least one edge incident with $x$ is not contained in $E'(\chi^*)$. We distinguish two cases.

**Case 1:** There is a vertex $y \in X_i$, such that each edge incident with $y$ is contained in $E'(\chi^*)$**.** Consider the $c$-coloring $\chi'$ for $G'$ obtained from $\chi^*$ by flipping the color of vertex $x$ to $\chi^*(y)$. Since $x$ and $y$ have the exact same neighborhood by definition of $G'$ and are not adjacent, each edge incident with $x$ is contained in $E'(\chi')$. Consequently, $\chi'$ is a better $c$-coloring for $G'$ than $\chi^*$. This contradicts the fact that $\chi^*$ is an optimal $c$-coloring for $G'$.

**Case 2:** Each vertex of $X_i$ is incident with at least one edge that is not contained in $E'(\chi^*)$**.** Since $X_i$ is an independent set in $G'$, this directly implies that $E'(\chi^*)$ misses at least $|X_i| = n^2$ edges of $E'$. Hence, $E(\chi^*)$ contains less edges than $E'(\chi)$ and is thus not an optimal $c$-coloring for $G'$, a contradiction. ■

By the above, we know that $\chi^*$ contains all edges of $E' \setminus E$. Since $G' - V$ is a complete $c$-partite graph with $c$-partition $(X_1, \ldots, X_c)$, there is a bijection $\pi \colon [1, c] \to [1, c]$, such that for each $i \in [1, c]$, each vertex of $X_i$ receives color $\pi(i)$ under $\chi^*$. Moreover, since for each $j \in [3, c]$, each vertex $v \in V$ is adjacent with each vertex of $X_j$, each vertex of $V$ receives either color $\pi(1)$ or color $\pi(2)$ under $\chi^*$. For simplicity, we assume in the following that $\pi$ is the identity function, that is, for each $i \in [1, c]$, each vertex of $X_i$ receives color $i$ under $\chi^*$ and each vertex of $V$ receives either color 1 or color 2 under $\chi^*$. Let $A' := \{v \in V \mid \chi^*(v) = 1\}$ and

let $B' := \{v \in V \mid \chi^*(v) = 2\}$. Note that $|E'(\chi^*)| = |E' \setminus E| + |E(A', B')|$ and that $|E'(\chi)| = |E' \setminus E| + |E(A, B)|$. Hence, $\chi^*$ is a better $c$-coloring for $G'$ than $\chi$ if and only if $(A', B')$ is a better partition of $G$ than $(A, B)$. Since $\chi^*$ improves over $\chi$, this implies that $(A, B)$ is not an optimal partition for $G$. By assumption there is an optimal partition for $G$ having flip-distance at most $k$ with $(A, B)$. Hence, $I$ is a yes-instance of LS Max Cut. By the first direction, this further implies that $I'$ is a yes-instance of LS Max $c$-Cut if $\chi$ is not an optimal $c$-coloring for $G'$.

Note that this implies that there is an optimal $c$-coloring for $G'$ having flip-distance at most $k$ with $\chi$. Hence, the reduction is also correct for the permissive version of LS Max $c$-Cut. Recall that LS Max Cut does not admit an FPT-algorithm when parameterized by $k$, unless FPT = W[1] and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. Since $|V'| \in n^{\mathcal{O}(1)}$, this implies that even the permissive version of LS Max $c$-Cut does not admit an FPT-algorithm when parameterized by $k$, unless FPT = W[1] and cannot be solved in $f(k) \cdot |V'|^{o(k)}$ time for any computable function $f$, unless the ETH fails. □

### 4.1.3 Hardness for Related Partition Problems

Based on Lemma 4.5, we are also able to show hardness for a previously considered local search version of Min Bisection and Max Bisection [52]. In both these problems, the input is again an undirected graph $G$ and the goal is to find a balanced partition $(X, Y)$ of the vertex set of $G$ that minimizes (maximizes) the edges in $E(X, Y)$. Here, a partition $(X, Y)$ is *balanced* if the size of $X$ and the size of $Y$ differ by at most one. Due to the close relation to LS Max Cut, the proposed local neighborhood for these problems is also the $k$-flip neighborhood. The corresponding local search problems in which we ask for a better balanced partition of flip-distance at most $k$ are denoted by LS Min Bisection and LS Max Bisection, respectively. It was shown that both these local search problems can be solved in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time on restricted graph classes [52] but W[1]-hardness on general graphs was not shown so far.

**Corollary 4.7.** LS Min Bisection *and* LS Max Bisection *are* W[1]*-hard when parameterized by $k$ and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. This running time lower bound holds even for the permissive version of both problems and both permissive versions do not admit* FPT*-algorithms when parameterized by $k$, unless* FPT = W[1].

*Proof.* First, we show the statement for LS Max Bisection. Afterwards, we discuss how to obtain the similar intractability result for LS Min Bisection. Let $I :=$

$(G = (V, E), \omega, A, B, k)$ be an instance of LS Max Cut with $\omega(e) = 1$ for each edge $e \in E$, such that there is an optimal partition $(A', B')$ for $G$ with flip-distance at most $k$ with $(A, B)$ and let $n := |V|$. Due to Lemma 4.5, even under these restrictions LS Max Cut is W[1]-hard when parameterized by $k$ and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails.

We obtain an equivalent instance $I' := (G', X, Y, k')$ of LS Max Bisection, by simply adding a large independent set to $G$. That is, we obtain the graph $G' = (V', E')$ by adding a set $Z$ of $n + 2k$ isolated vertices to $G$, setting $X := A \cup Z_A$ and $Y := B \cup (Z \setminus Z_A)$ for some arbitrary vertex set $Z_A \subseteq Z$ of size $|B| + k$, and setting $k' := 2k$. Note that $G$ and $G'$ have the exact same edge set and that $X$ and $Y$ have the same size and contain at least $k$ vertices of $Z$ each. Intuitively, this ensures that we can perform an improving $k$-flip on the vertices of $V$ and afterwards end back at a balanced partition by flipping at most $k$ additional vertices of $Z$ to the smaller part of the resulting potentially not balanced partition.

Note that for each partition $(X', Y')$ of $G'$, $E'(X', Y') = E(X \cap V, Y \cap V)$. This directly implies that $(X, Y)$ is an optimal balanced partition for $G'$ if $(X \cap V, Y \cap V) = (A, B)$ is an optimal partition for $G$. Hence, $I'$ is a no-instance of LS Max Bisection if $I$ is a no-instance of LS Max Cut, since by assumption, $(A, B)$ is an optimal partition for $G$ if and only if $I$ is a no-instance of LS Max Cut. It thus remains to show that $I'$ is a yes-instance of LS Max Bisection if $I$ is a yes-instance of LS Max Cut. By the above, this then implies that $I'$ is a no-instance of LS Max Bisection if and only if $(X, Y)$ is an optimal balanced partition for $G'$.

Assume that $I$ is a yes-instance of LS Max Cut. This implies that $(A, B)$ is not an optimal partition of $G$. Let $(A', B')$ be an optimal partition of $G$. By assumption, we can assume that $(A', B')$ has flip-distance at most $k$ with $(A, B)$. Let $\widehat{X} := A' \cup Z_A$ and $\widehat{Y} := B' \cup (Z \setminus Z_A)$. Note that $(\widehat{X}, \widehat{Y})$ has flip-distance at most $k$ with $(X, Y)$ and is a better partition for $G'$ than $(X, Y)$. Still, $(\widehat{X}, \widehat{Y})$ might not be a balanced partition. But since $(\widehat{X}, \widehat{Y})$ has flip-distance at most $k$ with $(X, Y)$, the size of $\widehat{X}$ and the size of $\widehat{Y}$ differ by at most $k$. Hence, we can obtain a balances partition $(X', Y')$ for $G'$ by flipping at most $k$ vertices from $Z$ from the larger part of the partition to the smaller part. This is possible, since by construction both $X$ and $Y$ contain at least $k$ vertices of $Z$. Since $(X', Y')$ is obtained from $(\widehat{X}, \widehat{Y})$ by flipping only isolated vertices, $(X', Y')$ is also a better partition for $G'$ than $(X, Y)$ and has flip-distance at most $2k = k'$ with $(X, Y)$. Consequently, $I'$ is a yes-instance of LS Max Bisection if $I$ is a yes-instance of LS Max Cut.

Hence, $I'$ is a no-instance of LS Max Bisection if and only if $(X, Y)$ is an optimal balanced partition for $G'$. This implies that the reduction also works for the permissive version of LS Max Bisection. Recall that LS Max Cut is W[1]-hard

when parameterized by $k$ and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. Since $|V'| \in n^{\mathcal{O}(1)}$ and $k' \in \mathcal{O}(k)$, this implies that (i) the strict version of LS Max Bisection is W[1]-hard when parameterized by $k'$, (ii) the permissive version of LS Max Bisection does not admit an FPT-algorithm when parameterized by $k'$, unless FPT = W[1], and (iii) both versions of LS Max Bisection cannot be solved in $f(k) \cdot |V'|^{o(k')}$ time for any computable function $f$, unless the ETH fails.

The reduction to LS Min Bisection works analogously by not considering the graph $G'$ as the input graph, but the complement graph $G'' := (V', E'')$ of $G'$. Consequently, for each balanced partitions $(X', Y')$ of $G''$, $|E''(X', Y')| = |V'|^2/4 - |E'(X', Y')|$. In other words, $(X', Y')$ is a better partition for $G''$ than $(X, Y)$ if and only if $(X', Y')$ is a better partition for $G'$ than $(X, Y)$. Hence, the intractability results also hold for the strict and permissive versions of LS Min Bisection. □

Additionally, based on the close relation of Max 2-SAT and Max Cut, we can also transfer new hardness results to the strict and permissive version of local search for Max SAT with respect to the $k$-flip neighborhood. This problem was considered by Szeider [161] under the name of $k$-Flip Max Sat. Here, the input is a Boolean formula $F$ in CNF, an assignment $\tau$ of the variables of $F$, and an integer $k$, and we ask for an assignment $\tau'$ of the variables of $F$ for which $d_{\text{flip}}(\tau, \tau') \leq k$ and that satisfies more clauses of $F$ than $\tau$. Szeider [161] showed that (i) the strict version of $k$-Flip Max Sat is W[1]-hard when parameterized by $k$ even on formulas where each clause has size two and (ii) the permissive version of $k$-Flip Max Sat does not admit an FPT-algorithm when parameterized by $k$, unless FPT = W[1], even when each clause has size at most three and the formula is either Horn or anti-Horn.[1]

**Theorem 4.8.** *Even on formulas $F$ where each clause has size two and contains exactly one positive and one negative literal, both the strict and the permissive versions of $k$-Flip Max Sat cannot be solved in $f(k) \cdot |F|^{o(k)}$ time for any computable function $f$, unless the ETH fails, the strict version of $k$-Flip Max Sat is W[1]-hard when parameterized by $k$, and the permissive version of $k$-Flip Max Sat does not admit an FPT-algorithm when parameterized by $k$, unless FPT = W[1].*

*Proof.* We present a reduction from LS Max 2-Cut, for which the desired intractability results hold even for the permissive version due to Lemma 4.5. Let $I := (G = (V, E), \omega, \chi, k)$ be an instance of LS Max Cut with $\omega(e) = 1$ for each edge $e \in E$. We define a formula $F$ as follows: The variables of $F$ are exactly

[1]Here, a formula is Horn (anti-Horn), if each clause contains at most one positive (negative) literal.

the vertices of $V$ and for each edge $\{u, v\} \in E$, $F$ contains the clauses $\{u, \neg v\}$ and $\{\neg u, v\}$.

Let $\tau : V \to \{1, 2\}$ be a 2-coloring of $V$. We interpret $\tau$ as an assignment for $F$, where color 1 (2) represents the truth value "true" ("false"). Note that by construction, for each edge $\{u, v\} \in E$, $\tau$ satisfies at least one of the clauses $\{u, \neg v\}$ and $\{\neg u, v\}$. Moreover, $\tau$ satisfies both clauses $\{u, \neg v\}$ and $\{\neg u, v\}$ if and only if the edge $\{u, v\}$ is properly colored under $\tau$. This implies that $\tau$ satisfies $|E| + |E(\tau)|$ clauses of $F$. Consequently, an assignment $\tau'$ of $F$ satisfies more clauses of $F$ that the 2-coloring $\chi$ if and only if $|E(\tau')| > |E(\chi)|$. Hence, $I$ is a yes-instance of LS Max 2-Cut if and only if $(F, \chi, k)$ is a yes-instance of $k$-Flip Max Sat. □

Hence, in comparison to the hardness results presented by Szeider [161], Theorem 4.8 provides a tight ETH lower bound as well as hardness for formulas that are 2-Sat, Horn and anti-Horn simultaneously.

## 4.2 Parameterized Algorithms for LS Max *c*-Cut

In this section, we complement the running time lower bound of Theorem 4.6 by presenting an algorithm for LS Max $c$-Cut that runs in $\Delta^{\mathcal{O}(k)} \cdot c \cdot n$ time, where $\Delta$ denotes the maximum degree of the input-graph. Our algorithm for LS Max $c$-Cut follows a simple framework: Generate a collection of candidate sets $S$ that may improve the coloring if the vertices in $S$ flip their colors. For each such candidate set $S$, we only know that the colors of the vertices of $S$ change, but we do not yet know which new color the vertices receive. To answer this question, that is, to determine whether there is any coloring of $S$ that leads to an improving coloring, we present an algorithm based on dynamic programming.

We first describe the subroutine that we use to find a best coloring for a given candidate set $S$ of vertices to flip.

**Theorem 4.9.** *Let $G = (V, E)$ be a graph, let $\omega \colon E \to \mathbb{Q}$ be an edge-weight function, let $\chi$ be a $c$-coloring of $G$, and let $S \subseteq V$ be a set of size at most $k$. One can compute in $\mathcal{O}(3^k \cdot c \cdot k^2 + k \cdot \Delta(G))$ time a $c$-coloring $\chi'$ of $G$ such that $D_{\text{flip}}(\chi, \chi') \subseteq S$ and $\omega(E(\chi'))$ is maximal among all such colorings.*

*Proof.* We use dynamic programming. Initially, we compute for each vertex $v \in S$ and each color $i \in [1, c]$ the weight $\theta_v^i$ of edges between $v$ and vertices of $V \setminus S$ that do not receive color $i$ under $\chi$, that is, $\theta_v^i := \omega(\{\{v, w\} \in E \mid w \in N(v) \setminus S, \chi(w) \neq i\})$. Moreover, we compute the weight $\omega_S$ of all properly colored edges of $E(S, N[S])$

as $\omega_S := \omega(\{\{u,v\} \in E(S, N[S]) \mid \chi(u) \neq \chi(v)\})$. This can be done in $\mathcal{O}(c \cdot k + k \cdot \Delta(G))$ time.

The table $T$ has entries of type $T[S', c']$ for each vertex set $S' \subseteq S$ and each color $c' \in [1, c]$. Each entry $T[S', c']$ stores the maximum total weights of properly colored edges with at least one endpoint in $S'$ and no endpoint in $S \setminus S'$ such that the following holds:

1. the vertices in $S'$ have some color in $[1, c']$, and
2. every vertex $v \in V \setminus S$ has color $\chi(v)$.

We start to fill the dynamic programming table by setting $T[S', 1] := \sum_{v \in S'} \theta_v^1$ for each vertex set $S' \subseteq S$.

For each vertex set $S' \subseteq S$ and each color $c' \in [2, c]$, we set

$$T[S', c'] := \max_{S'' \subseteq S'} T[S' \setminus S'', c' - 1] + \omega(E(S'', S' \setminus S'')) + \sum_{v \in S''} \theta_v^{c'}.$$

Intuitively, to find the best way to assign colors of $[1, c']$ to the vertices of $S'$, we search for the best vertex set $S'' \subseteq S'$, assign color $c'$ to all vertices of $S''$, and find the best way to assign the colors of $[1, c' - 1]$ to the vertices of $S' \setminus S''$. The maximal improvement $\omega(E(\chi')) - \omega(E(\chi))$ for any $c$-coloring $\chi'$ with $D_{\text{flip}}(\chi, \chi') \subseteq S$ can then be found by evaluating $T[S, c] - \omega_S$: this term corresponds to the maximum total weight of properly colored edges we get when distributing the vertices of $S$ among all color classes minus the original weights when every vertex of $S$ sticks with its color under $\chi$. The corresponding $c$-coloring can be found via traceback.

The formal correctness proof is straightforward and thus omitted. Hence, it remains to show the running time. The dynamic programming table $T$ has $2^k \cdot c$ entries. Each of these entries can be computed in $\mathcal{O}(2^{|S'|} \cdot k^2)$ time. Consequently, all entries can be computed in $\mathcal{O}(\sum_{i=0}^{k} \binom{k}{i} \cdot 2^i \cdot c \cdot k^2) = \mathcal{O}(3^k \cdot c \cdot k^2)$ time in total. Hence, the total running time is $\mathcal{O}(3^k \cdot c \cdot k^2 + k \cdot \Delta(G))$. □

For LS Max Cut, if we enforce that each vertex of $S$ changes its color, the situation is even simpler: When given a set $S \subseteq V$ of $k$ vertices that must flip their colors, the best possible improvement can be computed in $\mathcal{O}(k \cdot \Delta(G))$ time, since every vertex of $S$ must replace its color with the unique other color.

Recall that the idea of our algorithms for LS Max Cut and LS Max $c$-Cut is to iterate over possible candidate sets of vertices that may flip their colors. With the next lemma we show that it suffices to consider those vertex sets that are connected in the input graph. That is, we show that the input graph is a candidate support graph (defined in Section 2.8).

**Lemma 4.10.** *Let $I := (G = (V, E), c, \omega, \chi, k)$ be an instance of* LS Max $c$-Cut. *Then, for every inclusion-minimal improving $k$-flip $\chi'$ for $\chi$, the vertex set $D_{\text{flip}}(\chi, \chi')$ is connected in $G$.*

*Proof.* Let $\chi'$ be an inclusion-minimal improving $k$-flip for $\chi$. Let $S' := D_{\text{flip}}(\chi, \chi')$ be the vertices $\chi$ and $\chi'$ do not agree on and let $\mathcal{C}$ denote the connected components in $G[S']$. We show that if there are at least two connected components in $\mathcal{C}$, then there is an improving $k$-neighbor $\widetilde{\chi}$ of $\chi$ with $D_{\text{flip}}(\chi, \widetilde{\chi}) \subsetneq D_{\text{flip}}(\chi, \chi')$. For each connected component $C \in \mathcal{C}$, let $E_C^+ := (E(C, V) \cap E(\chi')) \setminus E(\chi)$ denote the set of properly colored edges in $E(\chi') \setminus E(\chi)$ that have at least one endpoint in $C$ and let $E_C^- := (E(C, V) \cap E(\chi)) \setminus E(\chi')$ denote the set of properly colored edges in $E(\chi) \setminus E(\chi')$ that have at least one endpoint in $C$. Note that $E(\chi') \setminus E(\chi) = \sum_{C \in \mathcal{C}} E_C^+$ and that $E(\chi) \setminus E(\chi') = \sum_{C \in \mathcal{C}} E_C^-$. Hence, the improvement of $\chi'$ over $\chi$ is

$$\omega(E(\chi')) - \omega(E(\chi)) = \sum_{C \in \mathcal{C}} \omega(E_C^+) - \sum_{C \in \mathcal{C}} \omega(E_C^-) = \sum_{C \in \mathcal{C}} (\omega(E_C^+) - \omega(E_C^-))$$

Since $\chi'$ improves over $\chi$, this implies that there is at least one connected component $S \in \mathcal{C}$ with $\omega(E_S^+) - \omega(E_S^-) > 0$. Let $\widetilde{\chi}$ be the $c$-coloring of $G$ that agrees with $\chi$ on all vertices of $V \setminus S$ and agrees with $\chi'$ on all vertices of $S$. Hence, $\widetilde{\chi}$ is an improving $k$-neighbor of $\chi$ with $D_{\text{flip}}(\chi, \widetilde{\chi}) \subsetneq D_{\text{flip}}(\chi, \chi')$. □

This implies that the input graph of each LS Max $c$-Cut-instance is a candidate support graph. Hence, due to Theorem 2.18, we derive that LS Max $c$-Cut can be solved in $(e \cdot \Delta(G)^k \cdot (c-1)^k \cdot |I|^{\mathcal{O}(1)}$ time, where the $|I|^{\mathcal{O}(1)}$ factor is $\mathcal{O}(k^3 \cdot n)$. Moreover, for $c > 4$, we can improve upon this running time: Due to Theorem 4.9, we can determine whether there is a better coloring $\chi'$ with $D_{\text{flip}}(\chi, \chi') \subseteq S$ in $3^{|S|} \cdot c \cdot |S|^2 + |S| \cdot \Delta(G)$ time. Hence, Theorem 2.17 implies the following running times for LS Max $c$-Cut.

**Theorem 4.11.** *In general,* LS Max $c$-Cut *can be solved in $\mathcal{O}((3 \cdot e)^k \cdot (\Delta(G) - 1)^{k+1} \cdot c \cdot k^3 \cdot n)$ time. Moreover,* LS Max Cut *can be solved in $\mathcal{O}(e^k \cdot (\Delta(G) - 1)^{k+1} \cdot k^2 \cdot n)$ time and* LS Max 3-Cut *can be solved in $\mathcal{O}((2 \cdot e)^k \cdot (\Delta(G) - 1)^{k+1} \cdot k^2 \cdot n)$ time.*

**Hill-Climbing Algorithm** To obtain not only a single improvement of a given coloring but a $c$-coloring with a total weight of properly colored edges as high as possible, we introduce the following hill-climbing algorithm.

Given an initial coloring $\chi$, we set the initial value of $k$ to 1. In each step, we use the above-mentioned algorithm for LS Max $c$-Cut to search for an improving coloring in the $k$-flip neighborhood of the current coloring. Whenever the algorithm

finds an improving $k$-neighbor $\chi'$ for the current coloring $\chi$, the current coloring gets replaced by $\chi'$ and $k$ gets set back to one. If the current coloring is $k$-optimal, the value of $k$ is incremented and the algorithm continues to search for an improvement in the new $k$-flip neighborhood. This is done until a given time limit is reached.

**ILP Formulation.** In our experiments, we also use the following ILP formulation for Max $c$-Cut. For each vertex $v \in V$ and each color $i \in [1, c]$, we use a binary variable $x_{v,i}$ which is equal to one if and only if $\chi'(v) = i$. We further use for each edge $e \in E$ a binary variable $y_e$ to indicate whether $e$ is properly colored with respect to $\chi'$. Thus, for each edge $\{u, v\} \in E$, the variable $y_{\{}u, v\}$ is set to one if and only if for each color $i \in [1, c]$, $x_{u,i} = 0$ or $x_{v,i} = 0$. This is ensured by the constraint $x_{u,i} + x_{v,i} + y_{\{u,v\}}$.

maximize $\sum_{e \in E} y_e \cdot \omega(e)$ subject to

$$\sum_{i \in [1,c]} x_{v,i} = 1 \qquad \text{for each } v \in V$$

$$x_{u,i} + x_{v,i} + y_{\{u,v\}} \leq 2 \qquad \text{for each } \{u, v\} \in E \text{ with } \omega(\{u, v\}) > 0 \text{ and each } i \in [1, c]$$

$$x_{u,i} + \sum_{j \in [1,c] \setminus \{i\}} x_{v,j} - y_{\{u,v\}} \leq 1 \qquad \text{for each } \{u, v\} \in E \text{ with } \omega(\{u, v\}) < 0 \text{ and each } i \in [1, c]$$

$$x_{v,i} \in \{0, 1\} \qquad \text{for each } v \in V \text{ and each } i \in [1, c]$$

$$y_e \in \{0, 1\} \qquad \text{for each } e \in E$$

Note that by adding the additional constraint $\sum_{v \in V} x_{v,\chi(v)} \geq |V| - k$, the ILP searches for a best $c$-coloring of the input graph having flip-distance at most $k$ with some initial $c$-coloring $\chi$. In other words, by adding this single constraint, the ILP solves LS Max $c$-Cut instead of Max $c$-Cut.

## 4.3 Speedup Strategies

We now introduce several speedup strategies that we use in our implementation to avoid enumerating all candidate sets.

### 4.3.1 Upper Bounds

To prevent the algorithm from enumerating all possible connected subsets of size at most $k$, we use upper bounds to determine for any given connected subset $S'$ of size smaller than $k$, if $S'$ can possibly be extended to a set $S$ of size $k$ such that there is an improving $c$-coloring $\chi'$ for $G$ where $S$ is exactly the set of vertices $\chi$ and $\chi'$ do not agree on. If there is no such possibility, we prevent our algorithm from enumerating supersets of $S'$. With the next definition we formalize this concept.

**Definition 4.12.** Let $I := (G, c, \omega, \chi, k)$ be an instance of LS Max $c$-Cut and let $S'$ with $|S'| < k$ be a subset of vertices of $G$. A value $b(I, S')$ is an *upper bound* if for each $c$-coloring $\chi'$ of $G$, with $S' \subsetneq D_{\text{flip}}(\chi, \chi')$ and $d_{\text{flip}}(\chi, \chi') = k$,

$$b(I, S') \geq \omega(E(\chi')).$$

In our implementation, we use upper bounds as follows: Given a set $S'$ we compute the value $b(I, S')$ and check if it is not larger than $\omega(E(\chi))$ for the current coloring $\chi$. If this is the case, we abort the enumeration of supersets of $S'$, otherwise, we continue.

We introduce two upper bounds; one for $c = 2$ and one for $c \geq 3$. To describe these upper bounds, we introduce the following notation: Given a vertex $v$ and a color $i$, we let $\omega_v^i := \omega(\{\{v, w\} \mid w \in N(v), \chi(w) \neq i\})$ denote the total weight of properly colored edges incident with $v$ if we change the color of $v$ to $i$ in in the $c$-coloring $\chi$. Thus, the term $\omega_v^i - \omega_v^{\chi(v)}$ describes the improvement obtained by changing only the color of $v$ to $i$. Furthermore, let $\omega_{\max} := \max_{e \in E} |\omega(e)|$ denote the maximum absolute edge weight.

**Upper Bound for $c = 2$.** Let $I$ be an instance of LS Max $c$-Cut with $c = 2$ and let $S'$ be a vertex set of size less than $k$. Since $c = 2$, we let $\overline{\chi(v)}$ denote the unique color distinct from $\chi(v)$ for each vertex $v$. For a vertex set $A \subseteq V$, let $\chi_A$ denote the coloring where $\chi_A(v) := \chi(v)$ for all $v \notin A$ and $\chi_A(v) = \overline{\chi(v)}$, otherwise. Intuitively, $\chi_A$ is the coloring resulting from $\chi$ when exactly the vertices in $A$ change their colors. For each vertex $v \in V \setminus S'$, we define $\alpha_v := \omega_v^{\overline{\chi(v)}} - \omega_v^{\chi(v)} + \beta_v$, where

$$\beta_v := \sum_{e \in E(v, S') \cap E(\chi)} 2 \cdot \omega(e) - \sum_{e \in E(v, S') \setminus E(\chi)} 2 \cdot \omega(e).$$

Intuitively, $\alpha_v - \beta_v$ is an upper bound for the improvement obtained when we choose to change only the color of $v$ to $\overline{\chi(v)}$. The term $\beta_v$ corresponds to the contribution of the edges between $v$ and the vertices of $S'$. In the definition of $\beta_v$, we take into account

the edges between $v$ and $S'$ that are falsely counted twice, once when extending $\chi_{S'}$ with $v$ and a second time in the term $\omega_v^{\overline{\chi(v)}} - \omega_v^{\chi(v)}$. Hence, $\alpha_v$ is the improvement over the coloring $\chi_{S'}$ obtained by changing only the color of $v$. Let $Y \subseteq V \setminus S'$ be the $k - |S'|$ vertices from $V \setminus S'$ with largest $\alpha_v$-values. We define the upper bound by

$$b_{c=2}(I, S') := \omega(E(\chi_{S'})) + \underbrace{\sum_{v \in Y} \alpha_v}_{(1)} + \underbrace{2\binom{k - |S'|}{2}\omega_{\max}}_{(2)}.$$

Recall that the overall goal is to find a set $X$ such that changing the colors of $S' \cup X$ results in a better coloring. The summand (1) corresponds to an overestimation of all weights of edges incident with exactly one vertex of $X$ by fixing the falsely counted edges between $X$ and $S'$ due to the included $\beta_v$ summands. The summand (2) corresponds to an overestimation of the weight of properly colored edges with both endpoints in $X$. We next show that $b_{c=2}$ is in fact an upper bound.

**Proposition 4.13.** *If $c = 2$, then $b_{c=2}(I, S')$ is an upper bound.*

*Proof.* Let $\chi'$ be a coloring with $S' \subsetneq D_{\text{flip}}(\chi, \chi')$ and $d_{\text{flip}}(\chi, \chi') = k$, and let $X := D_{\text{flip}}(\chi, \chi') \setminus S'$. We show that $\omega(E(\chi')) \leq b_{c=2}(I, S')$. To this end, we consider the coloring $\chi_{S'}$ that results from $\chi$ when exactly the vertices in $S'$ change their colors and analyze how $\omega(E(\chi'))$ differs from $\omega(E(\chi_{S'}))$.

$$\begin{aligned}\omega(E(\chi')) = \ & \omega(E(\chi_{S'})) + \underbrace{\sum_{v \in X} (\omega_v^{\overline{\chi(v)}} - \omega_v^{\chi(v)})}_{(1)} \\ & + \underbrace{\sum_{\substack{e \in E(X, S') \\ e \in E(\chi)}} 2 \cdot \omega(e) - \sum_{\substack{e \in E(X, S') \\ e \notin E(\chi)}} 2 \cdot \omega(e)}_{(2)} \\ & + \underbrace{\sum_{\substack{e \in E(X) \\ e \in E(\chi)}} 2 \cdot \omega(e) - \sum_{\substack{e \in E(X) \\ e \notin E(\chi)}} 2 \cdot \omega(e)}_{(3)}\end{aligned}$$

By adding (1) to $\omega(E(\chi_{S'}))$, we added the weight of all properly colored edges if only $v$ changes its color for every vertex $v \in X$. The difference between $\omega(E(\chi'))$

and $\omega(E(\chi_{S'})) + (1)$ then consists of all edge-weights that were falsely counted in (1) since both endpoints were moved. To compensate this, the summand (2) and (3) need to be added. Summand (2) corresponds to falsely counted edges with one endpoint in $X$ and one endpoint in $S'$, while (3) corresponds to falsely counted edges with both endpoints in $X$. Observe that every falsely counted edge weight was counted for both of its endpoints within $\omega(E(\chi_{S'})) + (1)$. Thus, each edge weight in (2) and (3) needs to be multiplied by 2.

Note that (3) is upper bounded by $2 \cdot \binom{k-|S'|}{2} \cdot \omega_{\max}$. Furthermore, note that $(1) + (2) = \sum_{v \in X} \alpha_v$. Recall that $Y$ consists of the $k - |S'|$ vertices from $V \setminus S'$ with largest $\alpha_v$-values. Hence, $(1) + (2) \leq \sum_{v \in Y} \alpha_v$. This implies that $\omega(E(\chi')) \leq b_{c=2}(I, S')$. Consequently, $b_{c=2}(I, S')$ is an upper bound. □

**Upper Bound for $c \geq 3$.** We next present an upper bound $b_{c\geq 3}$ that works for the case where $c \geq 3$. Recall that the upper bound $b_{c=2}$ relies on computing $\omega(E(\chi_{S'}))$, where $\chi_{S'}$ is the coloring resulting from $\chi$ when exactly the vertices in $S'$ change their colors. This was possible since for $c = 2$, there is only one coloring for which the flip with $\chi$ is exactly $S'$. In case of $c \geq 3$, each vertex in $S'$ has $c - 1 \geq 2$ options to change its color. Our upper bound $b_{c\geq 3}$ consequently contains a summand $b(S')$ that overestimates the edge weights when only the vertices in $S'$ change their colors.

To specify $b(S')$, we introduce the following notation: Given a vertex $v \in S'$ and a color $i$, we let

$$\theta_v^i := \omega(\{\{v, w\} \mid w \in N(v) \setminus S', \chi(w) \neq i\}).$$

Analogously to $\omega_v^i$, the value $\theta_v^i$ describes the weight of properly colored edges when changing the color of $v$ to $i$, but excludes all edges inside $S'$. We define the term

$$b(S') := \omega(E(\chi)) + \binom{|S'|}{2} \cdot \omega_{\max} - \sum_{\substack{e \in E(S') \\ e \in E(\chi)}} \omega(e) + \sum_{v \in S'} \left( \max_{i \neq \chi(v)} \theta_v^i - \theta_v^{\chi(v)} \right).$$

As mentioned above, for $b_{c\geq 3}$ the summand $b(S')$ replaces the summand $\omega(E(\chi_{S'}))$ which was used for $b_{c=2}$. Intuitively, the sum $\sum_{v \in S'} (\max_{i \neq \chi(v)} \theta_v^i - \theta_v^{\chi(v)})$ is an overestimation of the improvement for properly colored edges with exactly one endpoint in $S'$, the term $\binom{|S'|}{2} \cdot \omega_{\max}$ overestimates the properly colored edges inside $S'$, and the remaining terms overestimate the properly colored edges outside $S'$.

Analogously to $b_{c=2}$, for each vertex $v \in V \setminus S'$, we define a value $\alpha_v$ by $\alpha_v := \max_{i \neq \chi(v)}(\omega_v^i - \omega_v^{\chi(v)}) + \beta_v$ with

$$\beta_v := \sum_{e \in E(v,S')} 2 \cdot |\omega(e)|.$$

Again, let $Y \subseteq V \setminus S'$ be the $k - |S'|$ vertices with biggest $\alpha_v$-values of $V \setminus S'$. We define the upper bound by

$$b_{c\geq 3}(I, S') := b(S') + \sum_{v \in Y} \alpha_v + 2\binom{k - |S'|}{2} \cdot \omega_{\max}$$

and show that it is in fact an upper bound.

**Proposition 4.14.** *If $c \geq 3$, then $b_{c\geq 3}(I, S')$ is an upper bound.*

*Proof.* Let $\chi'$ be a coloring with $S' \subsetneq D_{\text{flip}}(\chi, \chi')$ and $d_{\text{flip}}(\chi, \chi') = k$, and let $X := D_{\text{flip}}(\chi, \chi') \setminus S'$. We show that $\omega(E(\chi')) \leq b_{c\geq 3}(I, S')$. To this end, let $\chi_{S'}$ denote the coloring that agrees with $\chi$ on all vertices of $V \setminus S'$ and that agrees with $\chi'$ on all vertices of $S'$. To show $\omega(E(\chi')) \leq b_{c\geq 3}(I, S')$ we analyze how $\omega(E(\chi'))$ differs from $\omega(E(\chi_{S'}))$.

$$\omega(E(\chi')) \leq \omega(E(\chi_{S'})) + \underbrace{\sum_{v \in X} (\omega_v^{\chi'(v)} - \omega_v^{\chi(v)})}_{(1)} + \underbrace{\sum_{e \in E(S',\, X)} 2 \cdot |\omega(e)|}_{(2)} + \underbrace{\sum_{e \in E(X)} 2 \cdot |\omega(e)|}_{(3)}$$

By adding (1) to $\omega(E(\chi_{S'}))$, we added the weight of all properly colored edges if only $v$ changes its color for every vertex $v \in X$. The difference between $\omega(E(\chi'))$ and $\omega(E(\chi_{S'})) + (1)$ then consists of all edge-weights that were falsely counted in (1) since both endpoints were moved. To compensate this, the summand (2) and (3) were added. Summand (2) overestimates the weight of falsely counted edges with one endpoint in $X$ and one endpoint in $S'$, while (3) overestimates the weight of falsely counted edges with both endpoints in $X$. Observe that every falsely counted edge weight may be counted for both of its endpoints within $\omega(E(\chi|_{S'})) + (1)$. Thus, each edge weight in (2) and (3) needs to be multiplied by 2.

Note that $(1)+(2) \leq \sum_{v \in X} \alpha_v \leq \sum_{v \in Y} \alpha_v$ and that $(3) \leq 2\binom{k-|S'|}{2} \cdot \omega_{\max}$. Therefore, it remains to show that $\omega(E(\chi_{S'})) \leq b(S')$. To this end, note that $\omega(E(\chi_{S'}))$ can be expressed by the sum of $\omega(E(\chi))$, improvement of the weight of properly colored edges inside $S'$ between $\chi$ and $\chi_{S'}$, and $\sum_{v \in S'}(\theta_v^{\chi'(v)} - \theta_v^{\chi(v)})$:

$$\begin{aligned}\omega(E(\chi_{S'})) = \ \omega(E(\chi)) + &\sum_{\substack{e \in E(S') \\ e \in E(\chi_{S'})}} \omega(e) - \sum_{\substack{e \in E(S') \\ e \in E(\chi)}} \omega(e) \\ &+ \sum_{v \in S'} (\theta_v^{\chi'(v)} - \theta_v^{\chi(v)}).\end{aligned}$$

Since $\binom{|S'|}{2} \cdot \omega_{\max}$ is at least as big as the sum of the weights of properly edges inside $S'$ under $\chi_{S'}$, we conclude $\omega(E(\chi_{S'})) \leq b(S')$. Hence, $b_{c \geq 3}$ is an upper bound. □

### 4.3.2 Prevention of Redundant Flips

We introduce further speed-up techniques that we used in our implementation of the hill-climbing algorithm. Roughly speaking, the idea behind these speed-up techniques is to exclude vertices that are not contained in an improving flip $D_{\text{flip}}(\chi, \chi')$ of any $k$-neighbor $\chi'$ of $\chi$. To this end, we introduce for each considered value of $k$ an *auxiliary vertex set* $V_k$ containing all remaining vertices that are potentially part of an improving flip of a $k$-neighbor of $\chi$. For each value of $k$, the set $V_k$ is initialized once with $V$, when we search for the first time for an improving $k$-neighbor.

It is easy to see that all vertices $x$ that are $(i,k)$-blocked for all $i \neq \chi(x)$ can be removed from $V_k$ if each edge of $G$ has weight 1. This also holds for general instances when considering an extension of the definition of $(i,k)$-blocked vertices for arbitrary weight functions. Moreover, whenever our algorithm has verified that a vertex $v$ is in no improving $k$-flip $D_{\text{flip}}(\chi, \chi')$, then we may remove $v$ from $V_k$.

Recall that we set the initial value of $k$ to one and increment $k$ if the current coloring $\chi$ is $k$-optimal. If at any time our algorithm replaces the current coloring $\chi$ by a better coloring $\chi'$, we set $k$ back to one and continue by searching for an improving $k$-neighbor of the new coloring $\chi'$, where $k$ again is incremented if necessary. Now, for each value of $k'$ that was already considered for a previous coloring, we only consider the remaining vertices of $V_{k'}$ together with vertices that have a small distance to the flip between $\chi'$ and the last previously encountered $(k'-1)$-optimal coloring. This idea is formalized by the next lemma.

**Lemma 4.15.** *Let $G = (V, E)$ be a graph, let $\omega\colon E \to \mathbb{Q}$ be an edge-weight function, and let $k$ be an integer. Moreover, let $\chi$ and $\chi'$ be $(k-1)$-optimal $c$-colorings of $G$ and let $v$ be a vertex within distance at least $k+1$ to each vertex of $D_{\text{flip}}(\chi, \chi')$. If there is no improving $k$-neighbor $\widehat{\chi}$ of $\chi$ with $v \in D_{\text{flip}}(\chi, \widehat{\chi})$, then there is no improving $k$-neighbor $\widetilde{\chi}$ of $\chi'$ with $v \in D_{\text{flip}}(\chi', \widetilde{\chi})$.*

*Proof.* We prove the lemma by contraposition. Let $\widetilde{\chi}$ be an improving $k$-neighbor of $\chi'$ with $v \in D_{\text{flip}}(\chi', \widetilde{\chi})$. We show that there is an improving $k$-neighbor $\widehat{\chi}$ of $\chi$ with $v \in D_{\text{flip}}(\chi, \widehat{\chi})$.

The $c$-coloring $\widehat{\chi}$ agrees with $\widetilde{\chi}$ on all vertices of $D_{\text{flip}}(\chi', \widetilde{\chi})$ and agrees with $\chi$ on all other vertices of $V$. Hence, $D_{\text{flip}}(\chi, \widehat{\chi})$ contains the vertex $v$. Moreover, $\widehat{\chi}$ and $\chi$ disagree on at most $d_{\text{flip}}(\chi', \widetilde{\chi}) \leq k$ positions which implies that $\widehat{\chi}$ is a $k$-neighbor of $\chi$. It remains to show that $\widehat{\chi}$ improves over $\chi$. To this end, we analyze the edge set $X \subseteq E$ of all edges with at least one endpoint in $D_{\text{flip}}(\chi', \widetilde{\chi})$. Consider the following claim about properly colored edges.

**Claim 3.** It holds that

*a)* $E(\widetilde{\chi}) \setminus X = E(\chi') \setminus X$ and $E(\widehat{\chi}) \setminus X = E(\chi) \setminus X$ ,

*b)* $E(\widetilde{\chi}) \cap X = E(\widehat{\chi}) \cap X$ and $E(\chi) \cap X = E(\chi') \cap X$.

*Proof of Claim.* *a)* Let $e$ be an edge of $E \setminus X$. Note that both endpoints of $e$ are elements of $V \setminus D_{\text{flip}}(\chi', \widetilde{\chi})$. Thus, the endpoints of $e$ have distinct colors under $\widetilde{\chi}$ if and only if they have distinct colors under $\chi'$. Consequently, $E(\widetilde{\chi}) \setminus X = E(\chi') \setminus X$. Furthermore, by definition of $\widehat{\chi}$, $D_{\text{flip}}(\chi, \widehat{\chi}) = D_{\text{flip}}(\chi', \widetilde{\chi})$, which implies that $E(\widehat{\chi}) \setminus X = E(\chi) \setminus X$.

*b)* Since $\chi'$ is $(k-1)$-optimal and $\widetilde{\chi}$ is an improving $k$-neighbor of $\chi'$, the set $D_{\text{flip}}(\chi', \widetilde{\chi})$ contains exactly $k$ vertices. Thus, we may assume by Lemma 4.10 that $D_{\text{flip}}(\chi', \widetilde{\chi})$ is connected. Consequently, each vertex of $D_{\text{flip}}(\chi', \widetilde{\chi})$ has distance at most $k-1$ from $v$, since $v$ is contained in $D_{\text{flip}}(\chi', \widetilde{\chi})$.

Let $e$ be an edge of $X$. Since each vertex of $D_{\text{flip}}(\chi', \widetilde{\chi})$ has distance at most $k-1$ from $v$, both endpoints of $e$ have distance at most $k$ from $v$. Together with the fact that $v$ has distance at least $k+1$ from $D_{\text{flip}}(\chi, \chi')$, this implies that no endpoint of $e$ is contained in $D_{\text{flip}}(\chi, \chi')$. Hence, $\chi$ and $\chi'$ agree on both endpoints of $e$. Consequently, the edge $e$ is properly colored under $\chi$ if and only if $e$ is properly colored under $\chi'$. This then implies that $E(\chi) \cap X = E(\chi') \cap X$.

By the definition of $\widehat{\chi}$ and the fact that $\chi$ and $\chi'$ agree on the endpoints of each edge of $X$, $\widehat{\chi}$ and $\widetilde{\chi}$ agree on the endpoints of each edge of $X$. This then implies that $E(\widetilde{\chi}) \cap X = E(\widehat{\chi}) \cap X$. ■

We next use Claim 3 to show that $\widehat{\chi}$ is an improving neighbor of $\chi$. Since $\widetilde{\chi}$ is an improving neighbor of $\chi'$ we have $\omega(E(\widetilde{\chi})) > \omega(E(\chi'))$, which implies

$$\omega(E(\widetilde{\chi}) \cap X) + \omega(E(\widetilde{\chi}) \setminus X) > \omega(E(\chi') \cap X) + \omega(E(\chi') \setminus X).$$

Together with Claim 3 *a*), we then have $\omega(E(\widetilde{\chi}) \cap X) > \omega(E(\chi') \cap X)$. Moreover, due to Claim 3 *b*), we have $\omega(E(\widehat{\chi}) \cap X) > \omega(E(\chi) \cap X)$. Finally, since $E(\widehat{\chi}) \setminus X = E(\chi) \setminus X$ by Claim 3 *a*), we may add the weights of all edges in $E(\widehat{\chi}) \setminus X$ to the left side of the inequality and the weight of all edges in $E(\chi) \setminus X$ to the right side. We end up with the inequality $\omega(E(\widehat{\chi})) > \omega(E(\chi))$ which implies that $\widehat{\chi}$ improves over $\chi$. □

We next describe how we exploit Lemma 4.15 in our implementation: We start with a coloring $\chi$ and search for improving $k$-neighbors of $\chi$ for increasing values of $k$ starting with $k = 1$. Whenever we find an improving neighbor $\chi'$ of $\chi$ we continue by searching for an improving neighbor $\chi''$ of $\chi'$ starting with $k = 1$ again. We use Lemma 4.15 as follows: if we want to find an improving $k$-neighbor for a $(k-1)$-optimal coloring $\chi'$, we take the last previously encountered $(k-1)$-optimal coloring $\chi$ and add only the vertices to $V_k$ that have distance at most $k$ from $D_{\text{flip}}(\chi, \chi')$, instead of setting $V_k$ back to $V$. This is correct since every vertex which is not in $V_k$, is not part of any improving $k$-flip of $\chi$ and therefore according to Lemma 4.15, the only vertices outside of $V_k$ that can possibly be in an improving $k$-flip of $\chi'$ are those with distance at most $k$ from $D_{\text{flip}}(\chi, \chi')$.

Next, we provide a further technique to identify vertices that can be removed from $V_k$. The idea behind this technique can be explained as follows: if a vertex can be excluded from $V_k$, then all equivalent vertices can also be excluded, where equivalence is defined as follows.

**Definition 4.16.** Let $G = (V, E)$ be a graph, let $\omega \colon E \to \mathbb{Q}$ be an edge-weight function. Two vertices $v$ and $w$ of $G$ are *weighted twins* if $N(v) \setminus \{w\} = N(w) \setminus \{v\}$ and $\omega(\{v, x\}) = \omega(\{w, x\})$ for each $x \in N(v) \setminus \{w\}$.

**Lemma 4.17.** *Let $G = (V, E)$ be a graph, let $\omega \colon E \to \mathbb{Q}$ be an edge-weight function, and let $k$ be an integer. Moreover, let $\chi$ be a $c$-coloring of $G$ and let $v$ and $w$ be weighted twins in $G$ with $\chi(v) = \chi(w)$. If there is no improving $k$-neighbor $\chi'$ of $\chi$ with $v \in D_{\text{flip}}(\chi, \chi')$, then there is no improving $k$-neighbor $\widetilde{\chi}$ of $\chi$ with $w \in D_{\text{flip}}(\chi, \widetilde{\chi})$.*

*Proof.* Assume towards a contradiction that there is an improving $k$-neighbor $\widetilde{\chi}$ of $\chi$ with $w \in D_{\text{flip}}(\chi, \widetilde{\chi})$. By assumption, $v \notin D_{\text{flip}}(\chi, \widetilde{\chi})$. Let $\chi'$ be the $c$-coloring that agrees with $\widetilde{\chi}$ on $V \setminus \{v, w\}$ and where $\chi'(v) := \widetilde{\chi}(w)$ and $\chi'(w) := \widetilde{\chi}(v) = \chi(v)$.

Recall that $\omega(\{v,x\}) = \omega(\{w,x\})$ for each $x \in N(v) \cap N(w)$ and that $N(v) \setminus \{w\} = N(w) \setminus \{v\}$. For each $x \in N(v) \cap N(w)$, let $E_x := \{\{v,x\},\{w,x\}\}$. Note that since $\widetilde{\chi}(v) \neq \widetilde{\chi}(w)$, at least one edge of $E_x$ is contained in $E(\widetilde{\chi})$. If both edges of $E_x$ are contained in $E(\widetilde{\chi})$, then $\chi'(x) = \widetilde{\chi}(x) \notin \{\chi'(v), \chi'(w)\}$ and thus both edges of $E_x$ are contained in $E(\chi')$. If only one edge of $E_x$ is contained in $E(\widetilde{\chi})$, then $E(\chi')$ contains exactly the other edge of $E_x$ since $\chi'(v) = \widetilde{\chi}(w)$, $\chi'(w) = \widetilde{\chi}(v)$, and $\chi'(x) = \widetilde{\chi}(x)$. Since the weight of both edges of $E_x$ are the same for each $x \in N(v) \cap N(w)$, we have $\omega(E(\chi')) = \omega(E(\widetilde{\chi}))$. Hence, $\chi'$ is an improving $k$-neighbor of $\chi$ with $v \in D_{\text{flip}}(\chi, \chi')$, a contradiction. □

Consequently, when our algorithm removes a vertex $v$ from $V_k$ for some $k$ because no improving $k$-neighbor $\chi'$ of $\chi$ contains $v$, then it also removes all weighted twins of $v$ with the same color as $v$ from $V_k$.

## 4.4 Implementation and Experimental Results

Our hill-climbing algorithm (LS) is implemented in JAVA/Kotlin and uses the graph library JGraphT. To enumerate all connected candidate sets, we use a JAVA implementation of a polynomial-delay algorithm for enumerating all connected induced subgraphs of a given size [108].

We used the graphs from the G-set benchmark[2], an established benchmark data set for Max $c$-Cut with $c \in \{2,3,4\}$ (and thus also for Max Cut) [15, 57, 120, 155, 171, 178]. The data set consists of 71 graphs with vertex-count between 800 and 20,000 and a density between 0.02% and 6%.

As starting solutions, we used the solutions computed by the MOH algorithm of Ma and Hao [120] for each graph of the G-set and each $c \in \{2,3,4\}$. For $c = 3$ and $c = 4$, these are the best known solutions for all graphs of the G-set. MOH is designed to quickly improve substantially on starting solutions but after a while progress stalls (we provide more details on this below). In contrast, our approach makes steady progress but is not as fast as MOH concerning the initial improvements, as preliminary experiments showed. Hence, we focus on evaluating the performance of LS as a post-processing for MOH by trying to improve their solutions quickly.

For one graph (g23) and each $c \in \{2,3,4\}$, there is a large gap between the value of the published coloring and the stated value of the corresponding coloring (for example, for $c = 3$, the published coloring has a value of 13 275 whereas it is stated that the coloring has a value of 17 168). To not exploit this gap in our

[2] `https://web.stanford.edu/~yyye/yyye/Gset/`

**Table 4.1:** The graphs from the G-set for which LS or ILP found an improved coloring, or for which we verified that MOH colorings are optimal (for $c = 3$). MOH shows the value of the published solutions of [120], LS and ILP show the best solution of our hill-climbing algorithm and any of the two ILP-runs, respectively. The best coloring is bold. Finally, UB shows the better upper bound computed during the two ILP-runs. For empty entries, no improved coloring was found. For bold UB entries, some found solution matches this upper bound, verifying its optimality.

| data | $\|V\|$ | $\|E\|$ | MOH | LS | ILP | UB |
|---|---|---|---|---|---|---|
| g11 | 800 | 1 600 | 669 | — | **671** | **671** |
| g12 | 800 | 1 600 | 660 | 661 | **663** | **663** |
| g13 | 800 | 1 600 | 686 | 687 | **688** | **688** |
| g15 | 800 | 4 661 | 3 984 | **3 985** | **3 985** | 4 442 |
| g24 | 2 000 | 19 990 | 17 162 | **17 163** | — | 19 989 |
| g25 | 2 000 | 19 990 | 17 163 | **17 164** | — | 19 989 |
| g26 | 2 000 | 19 990 | 17 154 | **17 155** | — | 19 989 |
| g27 | 2 000 | 19 990 | 4 020 | **4 021** | — | 9 840 |
| g28 | 2 000 | 19 990 | 3 973 | **3 975** | — | 9 822 |
| g31 | 2 000 | 19 990 | 4 003 | **4 005** | — | 9 776 |
| g32 | 2 000 | 4 000 | 1 653 | 1658 | **1 666** | 1 668 |
| g33 | 2 000 | 4 000 | 1 625 | 1628 | **1 636** | 1 640 |
| g34 | 2 000 | 4 000 | 1 607 | 1609 | **1 616** | 1 617 |
| g35 | 2 000 | 11 778 | 10 046 | **10 048** | — | 11 711 |
| g37 | 2 000 | 11 785 | 10 052 | **10 053** | **10 053** | 11 691 |
| g40 | 2 000 | 11 766 | 2 870 | **2 871** | — | 5 471 |
| g41 | 2 000 | 11 785 | 2 887 | **2 888** | — | 5 452 |
| g48 | 3 000 | 6 000 | **6 000** | — | — | **6 000** |
| g49 | 3 000 | 6 000 | **6 000** | — | — | **6 000** |
| g50 | 3 000 | 6 000 | **6 000** | — | — | **6 000** |
| g55 | 5 000 | 12 498 | 12 427 | 12 429 | **12 432** | 12 498 |
| g56 | 5 000 | 12 498 | 4 755 | **4 757** | — | 6 157 |
| g57 | 5 000 | 10 000 | 4 080 | 4092 | **4 103** | 4 154 |
| g59 | 5 000 | 29 570 | 7 274 | **7 276** | — | 14 673 |
| g61 | 7 000 | 17 148 | 6 858 | **6 861** | — | 8 728 |
| g62 | 7 000 | 14 000 | 5 686 | **5 710** | 5 706 | 5 981 |
| g63 | 7 000 | 41 459 | 35 315 | **35 318** | — | 41 420 |
| g64 | 7 000 | 41 459 | 10 429 | **10 437** | — | 20 713 |
| g65 | 8 000 | 16 000 | 6 489 | 6 512 | **6 535** | 6 711 |
| g66 | 9 000 | 18 000 | 7 414 | 7 442 | **7 443** | 7 843 |
| g67 | 10 000 | 20 000 | 8 088 | 8 116 | **8 141** | 9 080 |
| g70 | 10 000 | 9 999 | **9 999** | — | — | **9 999** |
| g72 | 10 000 | 20 000 | 8 190 | 8 224 | **8 244** | 9 166 |
| g77 | 14 000 | 28 000 | 11 579 | **11 632** | 11 619 | 13 101 |
| g81 | 20 000 | 40 000 | 16 326 | **16 392** | 16 374 | 18 337 |

evaluation, we only considered the remaining 70 graphs. These 70 graphs are of two types: 34 graphs are unit graphs (where each edge has weight 1) and 36 graphs are signed graphs (where each edge has either weight 1 or -1). For each of these graphs, we ran experiments for each $c \in \{2, 3, 4\}$ with a time limit of 30 minutes and the published MOH solution as initial solution. In addition to LS, for each instance we ran standard ILP-formulations for Max $c$-Cut (again for 30 minutes) using the Gurobi solver version 9.5, once without starting solution and once with the MOH solution as starting solution. Each run of an ILP provides both a best found solution and an upper bound on the maximum value of any $c$-coloring for the given instance. 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.

The ILP upper bounds verified the optimality of 22 MOH solutions. Thus, of

**Table 4.2:** The number of instances where LS or ILP found improved solutions. Column 'improvable' shows how many best known MOH colorings [120] *might* be suboptimal (as they do not meet the ILP upper bounds). Columns LS and ILP show how many of these solutions where improved by the respective approaches. Columns $I_1$, $I_2$, and $I_3$ show for how many instances the first improvement was found by LS within 10 seconds, between 10 and 60 seconds, and after more than 60 seconds, respectively.

| | improvable | $I_1$ | $I_2$ | $I_3$ | LS | ILP |
|---|---|---|---|---|---|---|
| unit $c = 2$ | 31 | 2 | 1 | 0 | 3 | 2 |
| unit $c = 3$ | 30 | 8 | 0 | 0 | 8 | 3 |
| unit $c = 4$ | 28 | 5 | 3 | 1 | 9 | 4 |
| signed $c = 2$ | 29 | 1 | 1 | 0 | 2 | 6 |
| signed $c = 3$ | 36 | 19 | 2 | 1 | 22 | 14 |
| signed $c = 4$ | 34 | 20 | 5 | 0 | 25 | 14 |
| sum | 188 | 55 | 12 | 2 | 69 | 43 |

the 210 instances, only 188 instances are interesting in the sense that LS or the ILP can find an improved solution. The upper bounds also verified the optimality of 8 further improved solutions found by LS or ILP.

In total, the ILP found better colorings than the MOH coloring for 43 of the 188 instances. In comparison, our hill-climbing algorithm was able to improve on the MOH solutions for 69 instances of the 188 instances. Table 4.1 gives the results for $c = 3$, showing those instances where the MOH coloring was verified to be optimal by the ILP or where LS or the ILP found an improved coloring. The full overview for $c \in \{2, 3, 4\}$ is shown in Tables 4.3 to 4.5.

Over all $c \in \{2, 3, 4\}$, on 35 instances, both LS and the ILP found improved colorings compared to the MOH coloring. For $c > 2$, both approaches find new record colorings. More precisely, for 23 instances, only the ILP found a new record coloring; for 6 instances, both approaches found a new record coloring, and for 38 instances only LS found a new record coloring. Thus, LS finds improvements also for very hard instances on which MOH provided the best known solutions so far.

The MOH solutions were obtained within a time limit of 30, 120, and 240 minutes for small, medium, and large instances, respectively. Each such run was repeated at least 10 times. The average time MOH took to find the best solution was 33% of the respective time limit. Hence, on average, after MOH found their best solution, in the remaining time (at least 20 minutes), MOH did not find any better solution. For all instances where LS was able to improve on the MOH solution, the average time to find the first improving flip was 15.17 seconds. Table 4.2 shows an overview on the

**Table 4.3:** The solutions of the best found $c$-coloring for any of the G-set graphs for $c = 2$. The column MOH shows the value of the published solutions of Ma and Hao [120].

| data | n | m | MOH | LS | ILP | UB | $ILP_1$ | $UB_1$ | $ILP_2$ | $UB_2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| g1 | 800 | 19176 | **11624** | — | — | 16188 | — | 16188 | — | 16225 |
| g2 | 800 | 19176 | **11620** | — | — | 15915 | — | 15915 | — | 16254 |
| g3 | 800 | 19176 | **11622** | — | — | 15766 | — | 15766 | — | 16058 |
| g4 | 800 | 19176 | **11646** | — | — | 16059 | — | 16217 | — | 16059 |
| g5 | 800 | 19176 | **11631** | — | — | 16182 | — | 16182 | — | 16220 |
| g6 | 800 | 19176 | **2178** | — | — | 6387 | — | 6627 | — | 6387 |
| g7 | 800 | 19176 | **2006** | — | — | 6225 | — | 6339 | — | 6225 |
| g8 | 800 | 19176 | **2005** | — | — | 6209 | — | 6209 | — | 6215 |
| g9 | 800 | 19176 | **2054** | — | — | 6106 | — | 6106 | — | 6379 |
| g10 | 800 | 19176 | **2000** | — | — | 6315 | — | 6315 | — | 6322 |
| g11 | 800 | 1600 | **564** | — | — | **564** | — | **564** | — | **564** |
| g12 | 800 | 1600 | **556** | — | — | **556** | — | **556** | — | **556** |
| g13 | 800 | 1600 | **582** | — | — | **582** | — | **582** | — | **582** |
| g14 | 800 | 4694 | **3064** | — | — | 3158 | — | 3158 | — | 3158 |
| g15 | 800 | 4661 | **3050** | — | — | 3139 | — | 3148 | — | 3139 |
| g16 | 800 | 4672 | **3052** | — | — | 3144 | — | 3144 | — | 3148 |
| g17 | 800 | 4667 | **3047** | — | — | 3144 | — | 3144 | — | 3153 |
| g18 | 800 | 4694 | **992** | — | — | 1137 | — | 1140 | — | 1137 |
| g19 | 800 | 4661 | **906** | — | — | 1044 | — | 1046 | — | 1044 |
| g20 | 800 | 4672 | **941** | — | — | 1069 | — | 1074 | — | 1069 |
| g21 | 800 | 4667 | **931** | — | — | 1072 | — | 1074 | — | 1072 |
| g22 | 2000 | 19990 | **13359** | — | — | 17677 | — | 17888 | — | 17677 |
| g24 | 2000 | 19990 | **13337** | — | — | 17905 | — | 17905 | — | 18070 |
| g25 | 2000 | 19990 | **13340** | — | — | 17993 | — | 17993 | — | 18049 |
| g26 | 2000 | 19990 | **13328** | — | — | 17744 | — | 18236 | — | 17744 |
| g27 | 2000 | 19990 | **3341** | — | — | 8273 | — | 8394 | — | 8273 |
| g28 | 2000 | 19990 | **3298** | — | — | 7505 | — | 8194 | — | 7505 |
| g29 | 2000 | 19990 | **3405** | — | — | 7594 | — | 7594 | — | 8382 |
| g30 | 2000 | 19990 | **3413** | — | — | 8033 | — | 8033 | — | 8354 |
| g31 | 2000 | 19990 | **3310** | — | — | 7688 | — | 8253 | — | 7688 |
| g32 | 2000 | 4000 | **1410** | — | — | **1410** | — | **1410** | — | **1410** |
| g33 | 2000 | 4000 | **1382** | — | — | **1382** | — | **1382** | — | **1382** |
| g34 | 2000 | 4000 | **1384** | — | — | **1384** | — | **1384** | — | **1384** |
| g35 | 2000 | 11778 | **7687** | — | — | 8985 | — | 9242 | — | 8985 |
| g36 | 2000 | 11766 | **7680** | — | — | 9125 | — | 9199 | — | 9125 |
| g37 | 2000 | 11785 | **7691** | — | — | 9056 | — | 9056 | — | 9265 |
| g38 | 2000 | 11779 | **7688** | — | — | 8923 | — | 8923 | — | 9130 |
| g39 | 2000 | 11778 | **2408** | — | — | 3214 | — | 3254 | — | 3214 |
| g40 | 2000 | 11766 | **2400** | — | — | 3157 | — | 3157 | — | 3218 |
| g41 | 2000 | 11785 | **2405** | — | — | 3196 | — | 3196 | — | 3199 |
| g42 | 2000 | 11779 | **2481** | — | — | 3228 | — | 3228 | — | 3229 |
| g43 | 1000 | 9990 | **6660** | — | — | 8055 | — | 8264 | — | 8055 |
| g44 | 1000 | 9990 | **6650** | — | — | 8228 | — | 8228 | — | 8287 |
| g45 | 1000 | 9990 | **6654** | — | — | 8146 | — | 8182 | — | 8146 |
| g46 | 1000 | 9990 | **6649** | — | — | 8148 | — | 8168 | — | 8148 |
| g47 | 1000 | 9990 | **6657** | — | — | 8075 | — | 8146 | — | 8075 |
| g48 | 3000 | 6000 | **6000** | — | — | **6000** | — | **6000** | — | **6000** |
| g49 | 3000 | 6000 | **6000** | — | — | **6000** | — | **6000** | — | **6000** |
| g50 | 3000 | 6000 | **5880** | — | — | **5880** | — | **5880** | — | **5880** |
| g51 | 1000 | 5909 | **3848** | — | — | 3992 | — | 4160 | — | 3992 |
| g52 | 1000 | 5916 | **3851** | — | — | 4095 | — | 4111 | — | 4095 |
| g53 | 1000 | 5914 | **3850** | — | — | 3971 | — | 3971 | — | 4090 |
| g54 | 1000 | 5916 | **3852** | — | — | 4073 | — | 4073 | — | 4082 |
| g55 | 5000 | 12498 | **10299** | — | — | 11692 | — | 11692 | — | 11755 |
| g56 | 5000 | 12498 | **4016** | — | — | 5378 | — | 5378 | — | 5418 |
| g57 | 5000 | 10000 | **3494** | — | — | **3494** | — | **3494** | — | **3494** |
| g58 | 5000 | 29570 | 19289 | **19290** | **19290** | 24833 | — | 24833 | **19290** | 25197 |
| g59 | 5000 | 29570 | **6086** | — | — | 10372 | — | 10372 | — | 10577 |
| g60 | 7000 | 17148 | **14190** | — | — | 16528 | — | 16547 | — | 16528 |
| g61 | 7000 | 17148 | **5797** | — | — | 8144 | — | 8144 | — | 8199 |
| g62 | 7000 | 14000 | 4868 | 4870 | **4872** | **4872** | **4872** | **4872** | **4872** | **4872** |
| g63 | 7000 | 41459 | 27033 | **27037** | — | 35046 | — | 35046 | — | 35703 |
| g64 | 7000 | 41459 | 8747 | **8748** | — | 15137 | — | 15137 | — | 15929 |
| g65 | 8000 | 16000 | 5560 | — | **5562** | 5568 | — | 5568 | **5562** | 5568 |
| g66 | 9000 | 18000 | 6360 | — | **6364** | 6368 | **6364** | 6368 | **6364** | 6369 |
| g67 | 10000 | 20000 | 6942 | — | **6948** | 6952 | — | 6957 | **6948** | 6952 |
| g70 | 10000 | 9999 | 9544 | 9551 | **9575** | 9714 | — | 9714 | **9575** | 9723 |
| g72 | 10000 | 20000 | 6998 | — | **7004** | 7013 | **7004** | 7013 | 7002 | 7014 |
| g77 | 14000 | 28000 | **9928** | — | — | 9948 | — | 9948 | — | 9951 |
| g81 | 20000 | 40000 | 14036 | — | **14044** | 14078 | — | 14078 | **14044** | 14080 |

**Table 4.4:** The solutions of the best found $c$-coloring for any of the G-set graphs for $c = 3$.

| data | n | m | MOH | LS | ILP | UB | $ILP_1$ | $UB_1$ | $ILP_2$ | $UB_2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| g1 | 800 | 19176 | **15165** | — | — | 19148 | — | 19148 | — | 19159 |
| g2 | 800 | 19176 | **15172** | — | — | 19160 | — | 19160 | — | 19160 |
| g3 | 800 | 19176 | **15173** | — | — | 19134 | — | 19134 | — | 19158 |
| g4 | 800 | 19176 | **15184** | — | — | 19117 | — | 19135 | — | 19117 |
| g5 | 800 | 19176 | **15193** | — | — | 19146 | — | 19146 | — | 19164 |
| g6 | 800 | 19176 | **2632** | — | — | 9441 | — | 9453 | — | 9441 |
| g7 | 800 | 19176 | **2409** | — | — | 9242 | — | 9253 | — | 9242 |
| g8 | 800 | 19176 | **2428** | — | — | 9228 | — | 9228 | — | 9230 |
| g9 | 800 | 19176 | **2478** | — | — | 9261 | — | 9275 | — | 9261 |
| g10 | 800 | 19176 | **2407** | — | — | 9233 | — | 9233 | — | 9267 |
| g11 | 800 | 1600 | 669 | — | **671** | **671** | **671** | **671** | **671** | 674 |
| g12 | 800 | 1600 | 660 | 661 | **663** | **663** | **663** | **663** | **663** | **663** |
| g13 | 800 | 1600 | 686 | 687 | **688** | **688** | **688** | **688** | **688** | **688** |
| g14 | 800 | 4694 | **4012** | — | — | 4497 | — | 4497 | — | 4510 |
| g15 | 800 | 4661 | 3984 | **3985** | **3985** | 4442 | — | 4442 | **3985** | 4474 |
| g16 | 800 | 4672 | **3990** | — | — | 4458 | — | 4465 | — | 4458 |
| g17 | 800 | 4667 | **3983** | — | — | 4413 | — | 4413 | — | 4457 |
| g18 | 800 | 4694 | **1207** | — | — | 1962 | — | 1962 | — | 1991 |
| g19 | 800 | 4661 | **1081** | — | — | 1833 | — | 1859 | — | 1833 |
| g20 | 800 | 4672 | **1122** | — | — | 1888 | — | 1888 | — | 1889 |
| g21 | 800 | 4667 | **1109** | — | — | 1827 | — | 1827 | — | 1875 |
| g22 | 2000 | 19990 | **17167** | — | — | 19989 | — | 19989 | — | 19989 |
| g24 | 2000 | 19990 | 17162 | **17163** | — | 19989 | — | 19989 | — | 19989 |
| g25 | 2000 | 19990 | 17163 | **17164** | — | 19989 | — | 19989 | — | 19989 |
| g26 | 2000 | 19990 | 17154 | **17155** | — | 19989 | — | 19990 | — | 19989 |
| g27 | 2000 | 19990 | 4020 | **4021** | — | 9840 | — | 9841 | — | 9840 |
| g28 | 2000 | 19990 | 3973 | **3975** | — | 9822 | — | 9827 | — | 9822 |
| g29 | 2000 | 19990 | **4106** | — | — | 9947 | — | 9948 | — | 9947 |
| g30 | 2000 | 19990 | **4117** | — | — | 9929 | — | 9929 | — | 9933 |
| g31 | 2000 | 19990 | 4003 | **4005** | — | 9776 | — | 9861 | — | 9776 |
| g32 | 2000 | 4000 | 1653 | 1658 | **1666** | 1668 | **1666** | 1668 | 1664 | 1670 |
| g33 | 2000 | 4000 | 1625 | 1628 | **1636** | 1640 | **1636** | 1640 | **1636** | 1640 |
| g34 | 2000 | 4000 | 1607 | 1609 | **1616** | 1617 | **1616** | 1617 | 1615 | 1618 |
| g35 | 2000 | 11778 | 10046 | **10048** | — | 11711 | — | 11711 | — | 11714 |
| g36 | 2000 | 11766 | **10039** | — | — | 11702 | — | 11702 | — | 11703 |
| g37 | 2000 | 11785 | 10052 | **10053** | **10053** | 11691 | — | 11753 | **10053** | 11691 |
| g38 | 2000 | 11779 | **10040** | — | — | 11703 | — | 11745 | — | 11703 |
| g39 | 2000 | 11778 | **2903** | — | — | 5457 | — | 5457 | — | 5551 |
| g40 | 2000 | 11766 | 2870 | **2871** | — | 5471 | — | 5471 | — | 5471 |
| g41 | 2000 | 11785 | 2887 | **2888** | — | 5452 | — | 5472 | — | 5452 |
| g42 | 2000 | 11779 | **2980** | — | — | 5551 | — | 5567 | — | 5551 |
| g43 | 1000 | 9990 | **8573** | — | — | 9985 | — | 9985 | — | 9988 |
| g44 | 1000 | 9990 | **8571** | — | — | 9957 | — | 9957 | — | 9980 |
| g45 | 1000 | 9990 | **8566** | — | — | 9983 | — | 9983 | — | 9986 |
| g46 | 1000 | 9990 | **8568** | — | — | 9983 | — | 9985 | — | 9983 |
| g47 | 1000 | 9990 | **8572** | — | — | 9966 | — | 9966 | — | 9983 |
| g48 | 3000 | 6000 | **6000** | — | — | **6000** | — | **6000** | — | **6000** |
| g49 | 3000 | 6000 | **6000** | — | — | **6000** | — | **6000** | — | **6000** |
| g50 | 3000 | 6000 | **6000** | — | — | **6000** | — | **6000** | — | **6000** |
| g51 | 1000 | 5909 | **5037** | — | — | 5708 | — | 5712 | — | 5708 |
| g52 | 1000 | 5916 | **5040** | — | — | 5703 | — | 5703 | — | 5726 |
| g53 | 1000 | 5914 | **5039** | — | — | 5694 | — | 5694 | — | 5746 |
| g54 | 1000 | 5916 | **5036** | — | — | 5667 | — | 5667 | — | 5682 |
| g55 | 5000 | 12498 | 12427 | 12429 | **12432** | 12498 | — | 12498 | **12432** | 12498 |
| g56 | 5000 | 12498 | 4755 | **4757** | — | 6157 | — | 6157 | — | 6176 |
| g57 | 5000 | 10000 | 4080 | 4092 | **4103** | 4154 | — | 4154 | **4103** | 4176 |
| g58 | 5000 | 29570 | **25195** | — | — | 29556 | — | 29556 | — | 29560 |
| g59 | 5000 | 29570 | 7274 | **7276** | — | 14673 | — | 14673 | — | 14678 |
| g60 | 7000 | 17148 | **17075** | — | — | 17148 | — | 17148 | — | 17148 |
| g61 | 7000 | 17148 | 6858 | **6861** | — | 8728 | — | 8728 | — | 8735 |
| g62 | 7000 | 14000 | 5686 | **5710** | 5706 | 5981 | — | 6033 | 5706 | 5981 |
| g63 | 7000 | 41459 | 35315 | **35318** | — | 41420 | — | 41420 | — | 41435 |
| g64 | 7000 | 41459 | 10429 | **10437** | — | 20713 | — | 20747 | — | 20713 |
| g65 | 8000 | 16000 | 6489 | 6512 | **6535** | 6711 | — | 6711 | **6535** | 6970 |
| g66 | 9000 | 18000 | 7414 | 7442 | **7443** | 7843 | — | 7843 | **7443** | 8246 |
| g67 | 10000 | 20000 | 8088 | 8116 | **8141** | 9080 | — | 9089 | **8141** | 9080 |
| g70 | 10000 | 9999 | **9999** | — | — | **9999** | — | **9999** | — | **9999** |
| g72 | 10000 | 20000 | 8190 | 8224 | **8244** | 9166 | — | 9243 | **8244** | 9166 |
| g77 | 14000 | 28000 | 11579 | **11632** | 11619 | 13101 | — | 13101 | 11619 | 13104 |
| g81 | 20000 | 40000 | 16326 | **16392** | 16374 | 18337 | — | 18515 | 16374 | 18337 |

**Table 4.5:** The solutions of the best found $c$-coloring for any of the G-set graphs for $c = 4$.

| data | n | m | MOH | LS | ILP | UB | $ILP_1$ | $UB_1$ | $ILP_2$ | $UB_2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| g1 | 800 | 19176 | **16803** | — | — | 19176 | — | 19176 | — | 19176 |
| g2 | 800 | 19176 | **16809** | — | — | 19176 | — | 19176 | — | 19176 |
| g3 | 800 | 19176 | **16806** | — | — | 19176 | — | 19176 | — | 19176 |
| g4 | 800 | 19176 | **16814** | — | — | 19176 | — | 19176 | — | 19176 |
| g5 | 800 | 19176 | **16816** | — | — | 19176 | — | 19176 | — | 19176 |
| g6 | 800 | 19176 | **2751** | — | — | 9544 | — | 9544 | — | 9583 |
| g7 | 800 | 19176 | **2515** | — | — | 9343 | — | 9430 | — | 9343 |
| g8 | 800 | 19176 | **2525** | — | — | 9397 | — | 9423 | — | 9397 |
| g9 | 800 | 19176 | **2585** | — | — | 9410 | — | 9410 | — | 9477 |
| g10 | 800 | 19176 | **2510** | — | — | 9380 | — | 9429 | — | 9380 |
| g11 | 800 | 1600 | **677** | — | — | **677** | — | **677** | — | **677** |
| g12 | 800 | 1600 | 664 | — | **665** | **665** | **665** | **665** | **665** | **665** |
| g13 | 800 | 1600 | **690** | — | — | **690** | — | **690** | — | **690** |
| g14 | 800 | 4694 | **4440** | — | — | 4670 | — | 4671 | — | 4670 |
| g15 | 800 | 4661 | **4406** | — | — | 4622 | — | 4622 | — | 4644 |
| g16 | 800 | 4672 | **4415** | — | — | 4630 | — | 4635 | — | 4630 |
| g17 | 800 | 4667 | **4411** | — | — | 4625 | — | 4636 | — | 4625 |
| g18 | 800 | 4694 | 1261 | **1262** | **1262** | 2122 | — | 2140 | **1262** | 2122 |
| g19 | 800 | 4661 | **1121** | — | — | 2045 | — | 2050 | — | 2045 |
| g20 | 800 | 4672 | **1168** | — | — | 2049 | — | 2049 | — | 2074 |
| g21 | 800 | 4667 | **1155** | — | — | 2023 | — | 2052 | — | 2023 |
| g22 | 2000 | 19990 | **18776** | — | — | 19990 | — | 19990 | — | 19990 |
| g24 | 2000 | 19990 | 18769 | **18772** | — | 19990 | — | 19990 | — | 19990 |
| g25 | 2000 | 19990 | 18775 | **18776** | — | 19990 | — | 19990 | — | 19990 |
| g26 | 2000 | 19990 | 18767 | **18770** | — | 19990 | — | 19990 | — | 19990 |
| g27 | 2000 | 19990 | 4201 | **4202** | — | 9928 | — | 9951 | — | 9928 |
| g28 | 2000 | 19990 | 4150 | **4157** | — | 9888 | — | 9888 | — | 9919 |
| g29 | 2000 | 19990 | 4293 | **4294** | — | 10010 | — | 10022 | — | 10010 |
| g30 | 2000 | 19990 | 4305 | **4308** | — | 10019 | — | 10019 | — | 10024 |
| g31 | 2000 | 19990 | 4171 | **4176** | — | 9910 | — | 9914 | — | 9910 |
| g32 | 2000 | 4000 | 1669 | 1671 | **1679** | **1679** | **1679** | **1679** | **1679** | **1679** |
| g33 | 2000 | 4000 | 1638 | 1640 | **1644** | **1644** | **1644** | **1644** | **1644** | **1644** |
| g34 | 2000 | 4000 | 1616 | 1617 | **1623** | **1623** | **1623** | **1623** | **1623** | 1625 |
| g35 | 2000 | 11778 | **11111** | — | — | 11775 | — | 11776 | — | 11775 |
| g36 | 2000 | 11766 | 11108 | — | **11109** | 11763 | — | 11763 | **11109** | 11764 |
| g37 | 2000 | 11785 | 11117 | **11118** | — | 11785 | — | 11785 | — | 11785 |
| g38 | 2000 | 11779 | 11108 | **11109** | — | 11778 | — | 11778 | — | 11778 |
| g39 | 2000 | 11778 | 3006 | **3007** | — | 5736 | — | 5794 | — | 5736 |
| g40 | 2000 | 11766 | 2976 | **2978** | — | 5665 | — | 5669 | — | 5665 |
| g41 | 2000 | 11785 | 2983 | **2986** | 2984 | 5751 | — | 5751 | 2984 | 5758 |
| g42 | 2000 | 11779 | 3092 | **3095** | — | 5787 | — | 5800 | — | 5787 |
| g43 | 1000 | 9990 | 9376 | **9377** | **9377** | 9990 | — | 9990 | **9377** | 9990 |
| g44 | 1000 | 9990 | **9379** | — | — | 9990 | — | 9990 | — | 9990 |
| g45 | 1000 | 9990 | 9376 | **9377** | **9377** | 9990 | — | 9990 | **9377** | 9990 |
| g46 | 1000 | 9990 | **9378** | — | — | 9990 | — | 9990 | — | 9990 |
| g47 | 1000 | 9990 | **9381** | — | — | 9990 | — | 9990 | — | 9990 |
| g48 | 3000 | 6000 | **6000** | — | — | **6000** | — | **6000** | — | **6000** |
| g49 | 3000 | 6000 | **6000** | — | — | **6000** | — | **6000** | — | **6000** |
| g50 | 3000 | 6000 | **6000** | — | — | **6000** | — | **6000** | — | **6000** |
| g51 | 1000 | 5909 | 5571 | **5572** | **5572** | 5871 | — | 5881 | **5572** | 5871 |
| g52 | 1000 | 5916 | **5584** | — | — | 5891 | — | 5891 | — | 5891 |
| g53 | 1000 | 5914 | **5574** | — | — | 5887 | — | 5887 | — | 5888 |
| g54 | 1000 | 5916 | **5579** | — | — | 5889 | — | 5889 | — | 5889 |
| g55 | 5000 | 12498 | **12498** | — | — | **12498** | — | **12498** | — | **12498** |
| g56 | 5000 | 12498 | 4931 | **4935** | — | 6213 | — | 6213 | — | 6213 |
| g57 | 5000 | 10000 | 4112 | 4132 | **4145** | 4220 | 4141 | 4305 | **4145** | 4220 |
| g58 | 5000 | 29570 | **27885** | — | — | 29569 | — | 29569 | — | 29570 |
| g59 | 5000 | 29570 | 7539 | **7546** | — | 14731 | — | 14731 | — | 14731 |
| g60 | 7000 | 17148 | **17148** | — | — | **17148** | — | **17148** | — | **17148** |
| g61 | 7000 | 17148 | 7110 | **7114** | — | 8748 | — | 8751 | — | 8748 |
| g62 | 7000 | 14000 | 5743 | 5758 | **5788** | 6534 | 5774 | 6534 | **5788** | 6541 |
| g63 | 7000 | 41459 | 39083 | **39089** | — | 41459 | — | 41459 | — | 41459 |
| g64 | 7000 | 41459 | 10814 | **10819** | — | 20775 | — | 20775 | — | 20792 |
| g65 | 8000 | 16000 | 6534 | 6561 | **6579** | 7256 | 6573 | 7256 | **6579** | 7349 |
| g66 | 9000 | 18000 | 7474 | 7495 | **7522** | 8497 | 7505 | 8497 | **7522** | 8500 |
| g67 | 10000 | 20000 | 8155 | 8185 | **8220** | 9299 | — | 9299 | **8220** | 9303 |
| g70 | 10000 | 9999 | **9999** | — | — | **9999** | — | **9999** | — | **9999** |
| g72 | 10000 | 20000 | 8264 | 8296 | **8337** | 9357 | — | 9357 | **8337** | 9376 |
| g77 | 14000 | 28000 | 11674 | **11712** | 11691 | 13296 | — | 13296 | 11691 | 13455 |
| g81 | 20000 | 40000 | 16470 | **16525** | 16485 | 19088 | — | 19088 | 16485 | 19580 |

**Table 4.6:** For each value of $k$, the number of instances for which an improving flip of size exactly $k$ was found.

| $k$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| unit $c = 2$ | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| unit $c = 3$ | 3 | 1 | 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| unit $c = 4$ | 2 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| signed $c = 2$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| signed $c = 3$ | 0 | 4 | 0 | 2 | 3 | 1 | 1 | 8 | 3 | 0 | 0 | 0 |
| signed $c = 4$ | 1 | 2 | 4 | 3 | 6 | 5 | 4 | 0 | 0 | 0 | 0 | 0 |
| sum | 6 | 7 | 11 | 10 | 10 | 8 | 5 | 8 | 3 | 0 | 0 | 1 |

**Table 4.7:** For each value of $k$, the number of instances for which the first improving flip that was found had size exactly $k$.

| $k$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| unit $c = 2$ | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| unit $c = 3$ | 4 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| unit $c = 4$ | 5 | 0 | 1 | 2 | 1 | 0 | 0 | 0 | 0 |
| signed $c = 2$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| signed $c = 3$ | 7 | 10 | 1 | 3 | 0 | 0 | 0 | 0 | 1 |
| signed $c = 4$ | 3 | 13 | 5 | 2 | 2 | 0 | 0 | 0 | 0 |
| sum | 20 | 26 | 9 | 9 | 3 | 1 | 0 | 0 | 1 |

number of improved instances and the time when LS found the first improvement. It is also interesting to see for which value of $k$ the first improvement was found (in other words, the smallest value $k$ such that the MOH solutions are not $k$-flip optimal). Table 4.7 shows for how many instances which value of $k$ was the smallest to obtain an improvement. On average, this value of $k$ was 3.39. Hence, it is indeed helpful to consider larger values of $k$ than the commonly used values of 1 or 2.

We summarize our main experimental findings as follows. First, parameterized local search can be used successfully as a post-processing for state-of-the-art heuristics for Max $c$-Cut, in many cases leading to new record solutions for $c > 2$ (see Tables 4.4 and 4.5). Second, the number of instances where an improvement was found is larger for LS than for the ILP approaches. Third, to find improved solutions, it is frequently necessary to explore $k$-flip neighborhoods for larger values of $k$ (see Tables 4.6 and 4.7). Finally, this can be done within an acceptable amount of time by using our algorithm for LS Max $c$-Cut and our speed-up strategies.

## 4.5 Concluding Remarks

In this chapter we analyzed LS Max $c$-Cut from both a theoretical and practical point of view. Form a negative point of view, we showed that both the strict and the permissive version of LS Max $c$-Cut cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. From a positive point of view, we presented an algorithm that solves these problems in $\Delta^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. Moreover, we implemented this algorithm and evaluated its performance as a post-processing for a state-of-the-art heuristic for Max $c$-Cut. Our experimental findings indicate that parameterized local search might be a promising technique in the design of local search algorithms and that its usefulness should be explored for further hard problems, in particular as post-processing for state-of-the-art heuristics to improve already good solutions.

**Open questions.** Our results in this chapter leave several questions open and give raise to new research directions. From a practical point of view, it would be interesting to consider a combined implementation of the MoH algorithm with our hill-climbing algorithm based on the $k$-flip neighborhood. In such a combined implementation, one could for example consider two different time limits $t_1$ and $t_2$: First, until time limit $t_1$ is reached, we let the MoH-algorithm run. Afterwards, we take the best found solution by MoH as starting solution for our hill-climbing algorithm which we then run until time limit $t_2$ is reached. It would be interesting to analyze the final solution quality with respect to these two time limits. In other words, it would be interesting to analyze when switching from MoH to our hill-climbing algorithm is promising. Such an approach could also be considered with respect to combinations of other state-of-the-art heuristics for Max $c$-Cut. For example for $c = 2$, that is, for Max Cut, one could consider analyzing the usefulness of our hill-climbing algorithm as a post-processing algorithm for algorithms like TS-UBQP [103] or TSHEA [177].

Regarding the practical evaluation of parameterized local search algorithms, we believe that some of the techniques introduced in this chapter might find successful applications also for other problems. In particular, the technique we introduced to prevent redundantly checking candidates (see Lemma 4.15) seems promising: We can ignore candidates that contain a vertex $v$ for which no vertex of distance $\mathcal{O}(k)$ has changed since the last time we verified that $v$ is not contained in any improving candidate. This technique was already successfully adapted in a local search solver for Weighted Vertex Cover [164][3] and we believe that it might find successful

[3]This bachelor's thesis was co-supervised by me.

application for problems like Max SAT [161] or Hitting Set. It seems possible to generalize or adapt the concept of candidate support graphs to formalize this technique for a large class of local search problems. In particular for Max SAT this seems possible, since flipping the truth value of a variable $v$ only has impact on other variables that share a clause with $v$.

From a theoretical point of view, one could ask for a smaller parameter in the basis of the worst-case running time. For example, one may ask whether we can replace the maximum degree in the basis by the $h$-index of the input graph, that is, if we can solve Max $c$-Cut in $h^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. In the aim of developing an algorithm with such a running time, one could for example branch into all possible ways to flip up to $k$ high-degree vertices. For each such branch, one then needs to find a coloring that improves over the initial coloring that only flips low-degree vertices. This would then necessitate solving a gap-version of Max $c$-Cut similar to the one introduced for LS Vertex Cover in Chapter 3.

Based on the W[1]-hardness results for LS Min Bisection and LS Max Bisection with respect to the search radius $k$, one might also consider revisiting the parameterized complexity of these problems with respect to $k$ plus some additional parameter $\ell$. So far, the only known algorithm for LS Min Bisection and LS Max Bisection run in FPT-time with respect to $k$ on graphs on bounded local treewidth [52]. These algorithms imply that LS Min Bisection and LS Max Bisection can be solved in $2^{\Delta^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$ time, but the existence of algorithms for these problems that run in $\Delta^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time are open. Based on the restriction that each part of the partition is equally-sized in any solution of these problems, the input graph is *not* a candidate support graph for these local search problems. In other words, one might need to consider candidates that are not connected in the input graph to achieve a better solution. One might also look into the complexity of local search for the even more general Graph Partitioning problem [7]. Here, Graph Partitioning asks for an $r$-coloring of the vertex set of a graph, such that the total number of properly colored edges is minimized, under the restriction that each color may be assigned to at most $c \cdot \lceil \frac{n}{r} \rceil$ vertices. Hence, Min Bisection is the special case of Graph Partitioning where $r = 2$ and $c = 1$. Graph Partitioning has applications in parallel computing, where the goal is to distribute computation units (the vertices) across $r$ processors so that as few communications (the edges of the graph) as possible are required between different processors [147]. Hence, improving over currently best heuristics for Graph Partitioning is highly important. Can hill-climbing algorithms based on scalable neighborhoods achieve this?

# Chapter 5

# Graph Clustering Problems under the Lens of Parameterized Local Search

Graph-based data clustering is a fundamental task with numerous applications [150]. Within this broad setting, we focus on the approach of modifying an input graph into a cluster graph (that is, a disjoint union of cliques) with as few edge modifications as possible. Herein, edges may be deleted or inserted, leading to the well-known Cluster Editing or Correlation Clustering problems [12, 14, 153]. Moreover, Cluster Deletion is the problem version where only edge deletions are allowed [153].

Cluster Deletion and Cluster Editing are highly relevant in practice, with application areas ranging from bioinformatics [14] to data mining [12] and psychology [163]. Unfortunately, Cluster Deletion and Cluster Editing are NP-hard [111, 153]. Therefore, efforts have been made to circumvent this hardness. One prominent approach is parameterized algorithms [20, 34, 53, 75, 81, 117]. Given the amount of research on parameterized algorithms and the practical relevance, it is no surprise that Cluster Editing was selected as the problem for the sixth installment of the parameterized implementation challenge PACE 2021 [99]. The results revealed the strength of local search for Cluster Editing: The top ten submissions in the heuristic track all involve local search. Moreover, the top three submissions always returned a solution less than 1.001 times larger than the optimal solution, that is, the relative error is below $10^{-3}$. This is in stark contrast to the best-known theoretical polynomial-time approximation having an approximation factor of 2.06 [32].

In this chapter we complement the results of the PACE 2021 heuristic track with

a theoretical study of the local search problems associated with Cluster Editing and Cluster Deletion. More precisely, we study the following question: Can we improve a given clustering of the input graph $G$ by moving at most $k$ vertices to different clusters? Here, a clustering can be viewed as an equivalence relation "which vertices will end up in the same cluster?" or, equivalently, a partition of the vertex set of $G$. Many local search algorithms submitted to PACE 2021 try to move vertices between clusters to improve their solution [13, 19, 70, 99, 160]. For Cluster Editing, we are free to move any vertex in any cluster (inserting missing edges within a cluster and deleting edges between clusters) while for Cluster Deletion we have to ensure that there are no missing edges within any cluster we create. The respective local search versions of the problems are called LS Cluster Editing and LS Cluster Deletion (see Section 5.1 for precise problem definitions).

**Related work.** Dörnfelder et al. [43] showed the W[1]-hardness of a local search version of Cluster Editing with a different local neighborhood: they search for a better solution by modifying at most $k$ edges in the given solution. Recently, Luo et al. [119] considered the closely related Dynamic Cluster Editing problem, in which a given clustering $\mathcal{C}$ for a graph $G$ has to be adapted while the graph $G$ changes dynamically, keeping the modification distance between successive clusterings low. In this setting, the main question is whether minor changes to $\mathcal{C}$ are sufficient to produce a good clustering for the (slightly) changed new graph. Hence, the old graph $G$ is actually irrelevant for the computational problem, there are only minor technical differences to LS Cluster Editing. Luo et al. [119] measure the distance to $\mathcal{C}$ by the number of vertices moving to a different cluster (they call this "matching distance")—the same measure we use. Luo et al. analyzed the parameterized complexity of Dynamic Cluster Editing with respect to the solution size $r$ (number of edges to modify in $G$) and the number $k$ of changes allowed to be done to the given solution: They obtained W[1]-hardness for each of the parameters $k$ and $r$ individually and fixed-parameter tractability with respect to the combined parameter $k + r$.

**Our results.** We consider the number $k$ of vertices allowed to be moved to different clusters, since this number can be expected to be small. In fact, $k$ assumes mostly one-digit values in PACE 2021 local search submissions. Again, this scalable neighborhood has a size of $n^{\mathcal{O}(k)}$, that is, LS Cluster Deletion and LS Cluster Editing admit trivial XP-algorithms when parameterized by $k$. In contrast, we show that LS Cluster Deletion and LS Cluster Editing are—like many other local search problems—W[1]-hard with respect to the search radius $k$. To cope with this hardness, we try to combine $k$ with different structural parameters $\ell$. If a parameter

combination $k + \ell$ allows for fixed-parameter tractability, then we aim for running times of the form $\ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. Thus, the resulting algorithms are expected to be very efficient in practice, which is particularly important since local search subroutines are called excessively in the solvers (see for example the PACE 2021 solvers [99]).

More precisely, we combine $k$ with the maximum degree $\Delta$, the maximum size of any clique in the given solution, or the cluster vertex deletion number cvd, that is, the number of vertices to remove to obtain a cluster graph.

In Section 5.2, we present some basic observations about both considered problems and derive algorithms for both problems that run in $(3 \cdot e)^k \cdot \Delta^{2k} \cdot n^{\mathcal{O}(1)}$ time. Intuitively, these algorithms are obtained by the observation that one can find a better solution by only moving vertex sets that are connected in the second power of the input graph.

In Section 5.3, we complement these algorithms with running time lower bounds for the strict version of LS Cluster Editing and the strict and the permissive version of LS Cluster Deletion. In particular, we show that the strict and the permissive version of LS Cluster Deletion do not admit an FPT-algorithm with respect to the sum of $k$ and the maximum cluster size, unless FPT = W[1]. For the strict version of LS Cluster Editing, we show W[1]-hardness (i) with respect to the sum of $k$, the maximum cluster size, and the degeneracy of the input graph $G$ and (ii) with respect to $k$ in the restricted case that the given clustering consists of only two clusters. The employed reductions also show that neither LS Cluster Editing nor LS Cluster Deletion can be solved in $f(k) \cdot n^{o(k)}$ time unless the ETH fails. This shows that in the above-mentioned algorithms the $\mathcal{O}(k)$ in the exponent cannot be replaced by $o(k)$.

In Section 5.4, we present algorithms for the permissive versions of both LS Cluster Deletion and LS Cluster Editing that run in $(\text{cvd} \cdot k)^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. As an intermediate result, we obtain an algorithm solving Cluster Deletion in $\text{cvd}^{\text{cvd}} \cdot n^{\mathcal{O}(1)}$ time.

## 5.1 Problem-Specific Notation

Let $G = (V, E)$ be a graph. A partition $\mathcal{C}$ of $V$ is called a *clustering of* $G$. Each vertex set $C$ of $\mathcal{C}$ is called a *cluster*. For a clustering $\mathcal{C}$, we denote by $\text{Cl}(\mathcal{C}) := \bigcup_{C \in \mathcal{C}} \binom{C}{2}$ the edges inside the clusters of $\mathcal{C}$. We say that $\mathcal{C}$ is a *proper clustering of* $G$, if $\text{Cl}(\mathcal{C}) \subseteq E$.

Let $\chi \colon V \to \mathbb{N}$ be a *coloring* of $V$. We say that color $i \in \mathbb{N}$ is *used by* $\chi$ if there is at least one vertex $v \in V$ with $\chi(v) = i$. For a coloring $\chi$, let $\mathcal{C}_\chi$ denote the clustering of $G$ where for each color $i$ used by $\chi$, $\chi^{-1}(i) = \{v' \in V \mid \chi(v') = i\}$ is a cluster of $\mathcal{C}_\chi$. We call $\chi$ a *cluster-coloring* of $\mathcal{C}_\chi$ and we call $\mathcal{C}_\chi$ the *clustering*

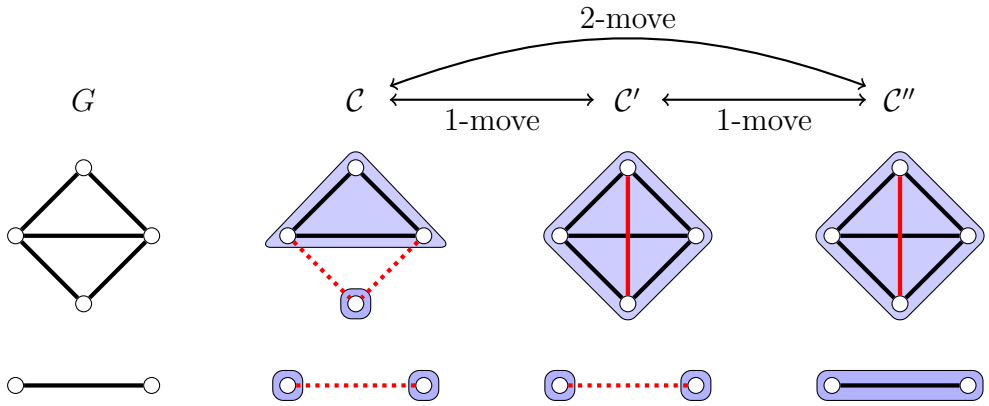



**Figure 5.1:** A graph $G$, three clusterings $\mathcal{C}$, $\mathcal{C}'$, and $\mathcal{C}''$ of $G$, and the move-distance between these clusterings. The blue polygons indicate the individual clusters of these three clusterings. The red edges indicate the necessary edge modifications to obtain the corresponding cluster graph form $G$. Dashed red edges are edges of $G$ that have to be removed and solid red edges are edges that have to be added to $G$ to obtain the respective clusterings.

of $\chi$. Note that each clustering has infinitely many cluster-colorings and that all these cluster-colorings are isomorphic. Here, two colorings $\chi$ and $\chi'$ are *isomorphic* if there is a bijection $f\colon \mathbb{N} \to \mathbb{N}$ such that $\chi = f \circ \chi'$.

Next, we define the considered scalable neighborhood for clusterings of a given graph, namely, the $k$-move neighborhood. This neighborhood is defined over the flip-distance between the respective cluster-colorings. To better illustrate the connection between the $k$-move neighborhood and the flip-distance between colorings, we may denote the flip-distance between two colorings in this chapter as the *move-distance*.

Let $\chi$ and $\chi'$ be colorings of $V$. We denote by $D_{\text{Move}}(\chi, \chi') := \{v \in V \mid \chi(v) \neq \chi'(v)\}$ the vertices that receive different colors under $\chi$ and $\chi'$. Moreover, we let $d_{\text{move}}(\chi, \chi') := |D_{\text{Move}}(\chi, \chi')|$ denote the move-distance between the colorings $\chi$ and $\chi'$. Note that $D_{\text{flip}}(\chi, \chi') = D_{\text{Move}}(\chi, \chi')$ and $d_{\text{flip}}(\chi, \chi') = d_{\text{move}}(\chi, \chi')$. Let $k \in \mathbb{N}$. Two colorings $\chi$ and $\chi'$ are *$k$-move neighbors* if $d_{\text{move}}(\chi, \chi') \leq k$. Analogously, two clusterings $\mathcal{C}$ and $\mathcal{C}'$ are *$k$-move neighbors* if there is a cluster-coloring $\chi$ of $\mathcal{C}$ and a cluster-coloring $\chi'$ of $\mathcal{C}'$ such that $\chi$ and $\chi'$ are $k$-move neighbors. We also denote by $d_{\text{move}}(\mathcal{C}, \mathcal{C}')$ the smallest integer $k$ such that $\mathcal{C}$ and $\mathcal{C}'$ are $k$-move neighbors. An example of $k$-move neighbors is depicted in Figure 5.1. Note that $d_{\text{move}}(\mathcal{C}, \mathcal{C}')$ can be computed in polynomial time [119].

For a clustering $\mathcal{C}$ we define $\text{cost}(\mathcal{C}) := |\text{Cl}(\mathcal{C}) \oplus E|$. Note that $\text{cost}_G(\mathcal{C}) =$

$\sum_{C\in\mathcal{C}} \text{cost}_G(C)$, where $\text{cost}_G(S) := |\binom{S}{2} \setminus E_G(S)| + \frac{|E_G(S,V\setminus S)|}{2}$ for each vertex set $S \subseteq V$. That is, $\text{cost}_G(S)$ counts the number of edges that are missing to make $S$ a clique plus the number of edges that need to be deleted to make $S$ a connected component. The number of edge deletion incident with some vertex of $S$ is further divided by 2 because we account for these edge deletions for both endpoints each such deleted edge. A clustering $\mathcal{C}'$ is *improving* over a clustering $\mathcal{C}$, if $\text{cost}_G(\mathcal{C}') < \text{cost}_G(\mathcal{C})$. If the graph $G$ is clear from the context, we may omit the subscript. In this chapter, we may also call a coloring $\chi$ improving over a coloring $\chi'$, if $\mathcal{C}_\chi$ is improving over $\mathcal{C}_{\chi'}$. An alternative evaluation of the quality of a clustering is the sum of values of the individual clusters. Here, for each vertex set $S \subseteq V$ the *value of cluster* $S$ is defined by $\text{val}(S) := 2 \cdot |E(S)| - |\binom{S}{2}|$.

**Observation 5.1.** *Let $\mathcal{C}$ and $\mathcal{C}'$ be clusterings of a graph $G$. Then* $\text{cost}(\mathcal{C}) - \text{cost}(\mathcal{C}') = -\sum_{C\in\mathcal{C}} \text{val}(C) + \sum_{C'\in\mathcal{C}'} \text{val}(C)$.

Hence, finding a clustering $\mathcal{C}$ of $G$ that minimizes $\text{cost}(\mathcal{C})$ is equivalent to finding a clustering $\mathcal{C}$ of $G$ that maximizes $\sum_{C\in\mathcal{C}} \text{val}(C)$.

We now formally define the considered problems in this chapter.

LS Cluster Deletion
**Input:** An undirected graph $G = (V, E)$, $k \in \mathbb{N}$, and a proper clustering $\mathcal{C}$ of $G$.
**Question:** Is there an improving $k$-move neighbor $\mathcal{C}'$ of $\mathcal{C}$ for $G$ such that $\mathcal{C}'$ is a proper clustering of $G$?

LS Cluster Editing
**Input:** An undirected graph $G = (V, E)$, $k \in \mathbb{N}$, and a clustering $\mathcal{C}$ of $G$.
**Question:** Is there an improving $k$-move neighbor $\mathcal{C}'$ of $\mathcal{C}$ for $G$?

**Observation 5.2.** *Let $G$ be a graph, let $k \in \mathbb{N}$, and let $\mathcal{C}$ and $\mathcal{C}'$ be clusterings of $G$ with $d_{\text{move}}(\mathcal{C}, \mathcal{C}') \leq k$. Then, $|\mathcal{C} \cap \mathcal{C}'| \geq |\mathcal{C}| - 2k$.*

## 5.2 Basic Observations

In this section, we observe several properties of the structure of improving solutions that we will use to present our algorithmic contributions. First, we show that the second power of the input graph is a candidate support graph for LS Cluster Deletion and for LS Cluster Editing. This then implies the existence of an

algorithm to enumerate a sufficient candidate collection for both problems in $(e \cdot \Delta(G)^2)^k \cdot n^{\mathcal{O}(1)}$ due to Lemma 2.16.

To this end, we show in following that a given clustering can trivially be improved over by moving a single vertex, if the clustering contains at least one cluster of diameter larger than 2. As we can check this case in polynomial time, we will then assume that in any given clustering in our instance of LS Cluster Editing, all clusters have diameter at most two. Note that for LS Cluster Deletion, the input clustering is proper and thus always fulfills this property.

**Observation 5.3.** *Let $G$ be a graph and let $\mathcal{C}$ be a clustering of $G$. If there is a cluster $C \in \mathcal{C}$, such that $G[C]$ has diameter at least three, then there is an improving 1-move neighbor $\mathcal{C}'$ of $\mathcal{C}$.*

*Proof.* Let $u, v \in C$ be two vertices of distance at least three in $G$. Thus, we have $N(u) \cap N(v) = \emptyset$. Assume without loss of generality that $|N(v) \cap C| \leq |N(u) \cap C|$.

We show that $\mathcal{C}' := (\mathcal{C} \setminus \{C\}) \cup \{C \setminus \{v\}, \{v\}\}$ is an improving 1-move neighbor of $\mathcal{C}$. Clearly, $\mathcal{C}'$ is a 1-move neighbor of $\mathcal{C}$ as only $v$ moves to a previously empty cluster. Moreover, this move costs $|N(v) \cap C|$ edge deletions but saves at least $|N(u) \cap C| + 1$ many edge insertions. Since $|N(v) \cap C| \leq |N(u) \cap C|$, it follows that $\mathcal{C}'$ is improving over $\mathcal{C}$. □

Next, we show that for LS Cluster Deletion and for LS Cluster Editing, $G^2$ is a candidate-support graph. Here, $G^2$ is the *second power of* $G$, that is, the graph on the vertex set of $G$, where two distinct vertices of $G^2$ are adjacent if and only if these vertices have distance at most two in $G$.

**Lemma 5.4.** *Let $G = (V, E)$ be a graph, let $\chi$ be a coloring of $V$, and let $k \geq 1$ be an integer. If there is an improving coloring $\chi'$ of $V$ with $d_{\text{move}}(\chi, \chi') \leq k$, then there is an improving coloring $\chi^*$ of $V$ with $d_{\text{move}}(\chi, \chi^*) \leq k$, such that $D_{\text{Move}}(\chi, \chi^*)$ is a connected vertex set in $G^2$. Moreover, if $\mathcal{C}_\chi$ and $\mathcal{C}_{\chi'}$ are proper clusterings of $G$, then $\mathcal{C}_{\chi^*}$ is a proper clustering of $G$.*

*Proof.* For each color $i$, let $V_i := \chi^{-1}(i)$ denote all vertices of $V$ receiving color $i$ under $\chi$. First, we show that if $G[V_i]$ has diameter more than two for some color $i$ used by $\chi$, then the statement holds. Let $i$ be a color used by $\chi$, such that $G[V_i]$ has diameter at least three. Then due to Observation 5.3, there is a coloring $\chi'$ of $V$ with $d_{\text{move}}(\chi, \chi') = 1$ and where $\mathcal{C}_{\chi'}$ is improving over $\mathcal{C}_\chi$. Hence, the statement holds, since $D_{\text{Move}}(\chi, \chi')$ contains a single vertex which implies that $G^2[D_{\text{Move}}(\chi, \chi')]$ is connected.

Thus, in the following, assume that for each color $i$ used by $\chi$, $G[V_i]$ has diameter at most two. Next, we show an even stronger property about the structure of the clusters $V_i$.

**Claim 4.** Let $i$ be a color used by $\chi$ and let $S \subseteq V_i$ be a set of vertices of size at most $k$ such that less than $\frac{|V_i \setminus S|}{2}$ vertices of $V_i \setminus S$ have neighbors in $S$. Then, there is a coloring $\chi'$ of $V$ with $d_{\text{move}}(\chi, \chi') \leq k$, such that $\mathcal{C}_{\chi'}$ improves over $\mathcal{C}_\chi$ and $G^2[D_{\text{Move}}(\chi, \chi')]$ is connected.

*Proof of Claim.* Consider the coloring $\chi'$ of $V$ that agrees with $\chi$ on all vertices of $V \setminus S$ and that assigns color $j$ to each vertex of $S$, where $j$ is a color not used by $\chi$. Hence, $\mathcal{C}_{\chi'} = (\mathcal{C}_\chi \setminus \{V_i\}) \cup \{S, V_i \setminus S\}$. This implies that the improvement of $\mathcal{C}_{\chi'}$ over $\mathcal{C}_\chi$ is

$$(|S| \cdot |V_i \setminus S| - |E(S, V_i \setminus S)|) - |E(S, V_i \setminus S)| = |S| \cdot |V_i \setminus S| - 2 \cdot |E(S, V_i \setminus S)|,$$

that is, the number of non-edges between vertices of $S$ and $V_i \setminus S$ minus the number of edges between vertices of $S$ and $V_i \setminus S$. Since less than $\frac{|V_i \setminus S|}{2}$ vertices of $V_i \setminus S$ have neighbors in $S$, $E(S, V_i \setminus S)$ has size less than $\frac{|S| \cdot |V_i \setminus S|}{2}$. Consequently, $\mathcal{C}_{\chi'}$ improves over $\mathcal{C}_\chi$, since $|S| \cdot |V_i \setminus S| - 2 \cdot |E(S, V_i \setminus S)|$ is strictly positive. Moreover, $d_{\text{move}}(\chi, \chi') = |S| \leq k$ and $G^2[S]$ is connected, since $V_i$ has diameter at most two in $G$. ■

Note that Claim 4 implies that we can assume in the following that for each color $i$ used by $\chi$ and each subset $S \subseteq V_i$, at least $\frac{|V_i \setminus S|}{2}$ vertices of $V_i \setminus S$ have neighbors in $S$.

In the following, for each coloring $\widehat{\chi}$ of $V$, we denote by $q_{\widehat{\chi}}$ the number of colors used by $\widehat{\chi}$ that are not used by $\chi$. Let $\chi'$ be a coloring of $V$ with $d_{\text{move}}(\chi, \chi') \leq k$ and where $\mathcal{C}_{\chi'}$ is improving over $\mathcal{C}_\chi$, such that $q_{\chi'}$ is maximized over all colorings $\widehat{\chi}$ of $V$ with $d_{\text{move}}(\chi, \widehat{\chi}) \leq k$ and where $\mathcal{C}_{\widehat{\chi}}$ is improving over $\mathcal{C}_\chi$. For each color $i$, let $V_i' := {\chi'}^{-1}(i)$, let $V_i^+ := V_i' \setminus V_i$ denote the vertices moved to color $i$, let $V_i^- := V_i \setminus V_i'$ denote the vertices moved out of color $i$, and let $V_i^\circ := V_i \cap V_i'$ denote the vertices receiving color $i$ under both $\chi$ and $\chi'$.

**Claim 5.** For each color $i$, $G^2[V_i^+ \cup V_i^-]$ is connected.

*Proof of Claim.* Note that for each color $i$, $G^2[V_i^-]$ is connected, since by assumption, $G[V_i]$ has diameter at most two, if $i$ is used by $\chi$ and $V_i^-$ is empty, otherwise. Next, we show that there is no color $i$, such that $V_i^\circ$ is empty and $V_i^+$ and $V_i^-$ are non-empty. Assume towards a contradiction that this is not the case and let $i$ be a color with $V_i^\circ = \emptyset$, $V_i^+ \neq \emptyset$, and $V_i^- \neq \emptyset$. Hence, each vertex receiving color $i$ under $\chi'$ is contained in $V_i^+ \subseteq D_{\text{Move}}(\chi, \chi')$. We define a coloring $\chi''$ of $V$ as follows:

The colorings $\chi'$ and $\chi''$ agree on all vertices of $V \setminus V_i^+$, and $\chi''$ assigns color $j$ to all vertices of $V_i^+$, where $j$ is an arbitrary color used by neither $\chi$ nor $\chi'$. Note that $V_i' = {\chi''}^{-1}(j)$, which implies that $\mathcal{C}_{\chi'} = \mathcal{C}_{\chi''}$. Consequently, $\mathcal{C}_{\chi''}$ improves over $\mathcal{C}_\chi$ and $d_{\text{move}}(\chi, \chi'') \leq k$, since each vertex of $V_i^+$ is contained in $D_{\text{Move}}(\chi, \chi')$. Moreover, since $j$ is a color not used by $\chi$ and color $i$ is used by $\chi$, $q_{\chi''} > q_{\chi'}$. This contradicts the fact that $\chi'$ maximizes $q_{\chi'}$ among all colorings $\widehat{\chi}$ of $V$ with $d_{\text{move}}(\chi, \widehat{\chi}) \leq k$ and where $\mathcal{C}_{\widehat{\chi}}$ is improving over $\mathcal{C}_\chi$.

Hence, we can assume in the following, that for each color $i$, $V_i^\circ$ is non-empty if both $V_i^+$ and $V_i^-$ are non-empty. To prove that for each color $i$, $G^2[V_i^+ \cup V_i^-]$ is connected, it is thus sufficient to show the following.

1. For each color $i$, $G^2[V_i^+]$ is connected and
2. for each color $i$, where all of the sets $V_i^+$, $V_i^\circ$, and $V_i^-$ are non-empty, there is a vertex $v \in V_i^\circ$, such that $v$ has at least one neighbor in both $V_i^+$ and $V_i^-$.

First, we show Item 1. Assume towards a contradiction that there is a color $i$, such that $G^2[V_i^+]$ is not connected. Since $G^2[V_i^+]$ is not connected, there are two vertices $u$ and $v$ of $V_i^+$ that have distance at least three in $G$. Then by the proof of Observation 5.3, there is some vertex $w \in \{u, v\}$, such that one can obtain a clustering $\mathcal{C}''$ that improves over $\mathcal{C}_{\chi'}$ by moving in $\mathcal{C}_{\chi'}$ the vertex $w$ into a new cluster containing only this vertex. Hence, there is a coloring $\chi''$ of $V$ with $D_{\text{Move}}(\chi', \chi'') = \{w\}$ and where $\chi''$ is a cluster-coloring of $\mathcal{C}''$. Note that this implies that $d_{\text{move}}(\chi, \chi'') \leq k$, since $w \in D_{\text{Move}}(\chi, \chi')$. Moreover, we can assume without loss of generality that the color $\chi''(w)$ is not used by $\chi$. Hence, $q_{\chi''} > q_{\chi'}$, a contradiction. Consequently, for each color $i$, $G^2[V_i^+]$ is connected.

Second, we show Item 2. Let $i$ be a color for which all the sets $V_i^+$, $V_i^\circ$, and $V_i^-$ are non-empty. We show that there is a vertex of $V_i^\circ$ which has a neighbor in $V_i^+$ and a neighbor in $V_i^-$. Recall that for each subset $S \subseteq V_i$ of size at most $k$, at least $\frac{|V_i \setminus S|}{2}$ vertices of $V_i \setminus S$ have neighbors in $S$. This holds in particular for $S = V_i^-$ and $V_i \setminus S = V_i^\circ$, since $V_i^- \subseteq D_{\text{Move}}(\chi, \chi')$ has size at most $k$. Hence, at least $\frac{|V_i^\circ|}{2}$ vertices of $V_i^\circ$ have neighbors in $V_i^-$.

To show that there is a vertex of $V_i^\circ$ which has a neighbor in $V_i^+$ and a neighbor in $V_i^-$, it thus suffices to show that more than $\frac{|V_i^\circ|}{2}$ vertices of $V_i^\circ$ have neighbors in $V_i^+$. Assume towards a contradiction that this is not the case. We show that there is a coloring $\chi''$ of $V$ such that $d_{\text{move}}(\chi, \chi'') \leq k$, $\mathcal{C}_{\chi''}$ improves over $\mathcal{C}_\chi$, and $q_{\chi''} > q_{\chi'}$. Consider the coloring $\chi''$ of $V$ that agrees with $\chi'$ on all vertices of $V \setminus V_i^+$ and that assigns color $j$ to each vertex of $V_i^+$, where $j$ is a color not used by $\chi$ or $\chi'$. Hence, $\mathcal{C}_{\chi''} = (\mathcal{C}_{\chi'} \setminus \{V_i^+ \cup V_i^\circ\}) \cup \{V_i^+, V_i^\circ\}$. This implies that the improvement of $\mathcal{C}_{\chi''}$

over $\mathcal{C}_{\chi'}$ is $(|V_i^+| \cdot |V_i^\circ| - |E(V_i^+, V_i^\circ)|) - |E(V_i^+, V_i^\circ)| = |V_i^+| \cdot |V_i^\circ| - 2 \cdot |E(V_i^+, V_i^\circ)|$, that is, the number of non-edges between vertices of $V_i^+$ and $V_i^\circ$ minus the number of edges between vertices of $V_i^+$ and $V_i^\circ$. Since at most $\frac{|V_i^\circ|}{2}$ vertices of $V_i^\circ$ have neighbors in $V_i^+$, $E(V_i^+, V_i^\circ)$ has size at most $\frac{|V_i^+| \cdot |V_i^\circ|}{2}$. Consequently, $\mathcal{C}_{\chi'}$ does not improve over $\mathcal{C}_{\chi''}$, since $|V_i^+| \cdot |V_i^\circ| - 2 \cdot |E(V_i^+, V_i^\circ)|$ is non-negative. This implies that $\mathcal{C}_{\chi''}$ improves over $\mathcal{C}_\chi$. Moreover, $d_{\text{move}}(\chi, \chi'') \leq k$, since $D_{\text{Move}}(\chi', \chi'') = V_i^+ \subseteq D_{\text{Move}}(\chi, \chi')$. Finally, $q_{\chi''} > q_{\chi'}$, since $j$ is a color not used by $\chi$ or $\chi'$, a contradiction. Hence, more than $\frac{|V_i^\circ|}{2}$ vertices of $V_i^\circ$ have neighbors in $V_i^+$ and at least $\frac{|V_i^\circ|}{2}$ vertices of $V_i^\circ$ have neighbors in $V_i^-$. This implies that there is at least one vertex of $V_i^\circ$ which has neighbors in both $V_i^+$ and $V_i^-$. ■

Based on Claim 5, we are finally ready to prove that there is a coloring $\chi^*$ of $V$ with $d_{\text{move}}(\chi, \chi^*) \leq k$ and where $\mathcal{C}_{\chi^*}$ improves over $\mathcal{C}_\chi$, such that $D_{\text{Move}}(\chi, \chi^*)$ is a connected vertex set in $G^2$. Let $\mathcal{S}$ denote the connected components of $G^2[D_{\text{Move}}(\chi, \chi')]$. For each connected component $S \in \mathcal{S}$, we denote by $X_S := \{\chi(v), \chi'(v) \mid v \in S\}$ the set of all colors affected by moving the vertices of $S$. Due to Claim 5, for each pair of distinct connected components $S$ and $S'$ of $\mathcal{S}$, $X_S$ and $X_{S'}$ are disjoint. In the following, we show that it is sufficient to move only the vertices of a single connected component of $\mathcal{S}$ to obtain a coloring $\chi^*$ for which $\mathcal{C}_{\chi^*}$ improves over $\mathcal{C}_\chi$. Note that $V_i = V_i'$ for each color $i \in \mathbb{N} \setminus \cup_{S \in \mathcal{S}} S$, which implies that $\text{cost}(V_i) = \text{cost}(V_i')$. Hence, the improvement of $\mathcal{C}_{\chi'}$ over $\mathcal{C}_\chi$ is

$$\begin{aligned}
\text{cost}(\mathcal{C}_\chi) - \text{cost}(\mathcal{C}_{\chi'}) &= \sum_{i \in \mathbb{N}} \text{cost}(V_i) - \sum_{i \in \mathbb{N}} \text{cost}(V_i') \\
&= \sum_{i \in \mathbb{N}} \text{cost}(V_i) - \text{cost}(V_i') \\
&= \sum_{S \in \mathcal{S}} \sum_{i \in S} \text{cost}(V_i) - \text{cost}(V_i').
\end{aligned}$$

Since this improvement is positive, there is at least one connected component $S \in \mathcal{S}$, such that $\sum_{i \in S} \text{cost}(V_i) - \text{cost}(V_i') > 0$. Let $\chi^*$ be the coloring of $V$ that agrees with $\chi$ on all vertices of $V \setminus S$ and that agrees with $\chi'$ on all vertices of $S$. For each color $i$, let $V_i^* := \chi^{*-1}(i)$. Note that $V_i^* = V_i$ for each color $i \in \mathbb{N} \setminus S$ and $V_i^* = V_i'$ for each color $i \in S$. Hence, the improvement of $\mathcal{C}_{\chi^*}$ over $\mathcal{C}_\chi$ is

$$\begin{aligned}
\text{cost}(\mathcal{C}_\chi) - \text{cost}(\mathcal{C}_{\chi^*}) &= \sum_{i \in \mathbb{N}} \text{cost}(V_i) - \sum_{i \in \mathbb{N}} \text{cost}(V_i^*) = \sum_{i \in \mathbb{N}} \text{cost}(V_i) - \text{cost}(V_i^*) \\
&= \sum_{i \in S} \text{cost}(V_i) - \text{cost}(V_i^*) = \sum_{i \in S} \text{cost}(V_i) - \text{cost}(V_i').
\end{aligned}$$

This improvement is positive by the choice of $S$. Since $D_{\text{Move}}(\chi, \chi^*) = S$ has size at most $k$ and is connected in $G^2$, the statement follows.

Note that if $\mathcal{C}_\chi$ and $\mathcal{C}_{\chi'}$ are proper clusterings of $G$, then all describe clustering that improves over $\mathcal{C}_\chi$ are also proper. Hence, if $\mathcal{C}_\chi$ and $\mathcal{C}_{\chi'}$ are proper clusterings of $G$, then $\mathcal{C}_{\chi^*}$ is a proper clustering of $G$. □

This implies that $G^2$ is a candidate support graph for both LS Cluster Deletion and LS Cluster Editing. Hence, Theorem 2.17 implies that we can search the $k$-move neighborhood in FPT-time with respect to $k$ and $\Delta(G)$, if we can determine efficiently for any given vertex set $S$ of size at most $k$, whether there is a better clustering one can obtain by moving only vertices of $S$.

In the following, we show that the latter task can be solved in time $3^{|S|} \cdot n^{\mathcal{O}(1)}$. To present our algorithm, we first define the notion of extensions of partial clusterings.

Let $G = (V, E)$ be a graph and let $\mathcal{C}$ be a clustering of $G$. Moreover, let $S \subseteq V$ and let $\mathcal{C}_S$ be a clustering of $G[S]$. We say that $\mathcal{C}$ *extends* $\mathcal{C}_S$ if for each cluster $C \in \mathcal{C}_S$ there is a cluster $C' \in \mathcal{C}$ with $C' \cap S = C$.

**Lemma 5.5.** *Let $G = (V, E)$ be a graph, let $\chi$ be a coloring of $V$, and let $S \subseteq V$ be a vertex set. One can find a best coloring $\chi'$ of $V$ fulfilling $D_{\text{Move}}(\chi, \chi') \subseteq S$ and find a best coloring $\chi''$ of $V$ fulfilling $D_{\text{Move}}(\chi, \chi'') \subseteq S$ and for which $\mathcal{C}_{\chi''}$ is a proper clustering of $G$ (if one exists) in time $3^{|S|} \cdot n^{\mathcal{O}(1)}$.*

*Proof.* We describe a dynamic program solving this task.

Let $\mathcal{C}_{V \setminus S}$ be the unique clustering of $G[V \setminus S]$ that is extended by $\mathcal{C}_\chi$. Fix an arbitrary ordering of the clusters of $\mathcal{C}_{V \setminus S}$ and let $C_i$ denote the $i$th cluster of $\mathcal{C}_{V \setminus S}$ according to this ordering. The dynamic programming table $T$ has entries of type $T[X, i]$ with $X \subseteq S$ and $i \in [0, |\mathcal{C}_{V \setminus S}|]$. For $X \subseteq S$ and $i \in [0, |\mathcal{C}_{V \setminus S}|]$, let $V_X^i$ denote the union of $X$ and the first $i$ clusters of $\mathcal{C}_{V \setminus S}$. The entry $T[X, i]$ stores the maximum value of any clustering of $G[V_X^i]$ that extends $\{C_j \mid 1 \leq j \leq i\}$.

To compute an entry $T[X, i]$, we iterate over all subsets $X'$ of $X$ and check for the best way to distribute the vertices of $X \setminus X'$ among the first $i-1$ clusters of $\mathcal{C}_{V \setminus S}$, while moving all vertices of $X'$ to cluster $C_i$.

Formally, for each $X \subseteq S$, we set $T[X, 0] := \text{OPT}(X)$, where $\text{OPT}(X)$ denotes the maximum value of any clustering of $G[X]$. For each $i \in [1, |\mathcal{C}_{V \setminus S}|]$ and each $X \subseteq S$, we set

$$T[X, i] := \max_{X' \subseteq X} T[X \setminus X', i-1] + \text{val}(C_i \cup X_i).$$

Note that to evaluate all base cases $T[X]$ of the dynamic programming table $T$, we have to compute for each $X \subseteq S$, the optimal values of any clustering of $G[X]$.

To this end, we describe an additional dynamic programming table $D$ that fulfills $D[X] = \mathrm{OPT}(X)$ for each $X \subset S$. Next, we describe how to compute the entries of the dynamic programming table $D$. Essentially, this dynamic programming table works the same way as the table $T$, except that we do not move the vertices of $X'$ into already existing clusters. Formally, we set $D[\emptyset] = 0$ and for each $X \subseteq S$, we set

$$D[X] := \max_{\substack{X' \subseteq X \\ X' \neq \emptyset}} D[X \setminus X'] + \mathrm{val}(X').$$

The maximum value of any sought clustering can then be found in $T[X, |\mathcal{C}_{V \setminus S}|]$. A corresponding clustering (and thus a sought coloring) can be found via traceback. Moreover, when searching only for a proper clustering, one has to only consider subsets $X' \subseteq X$ during the evaluation of the recurrence of $T$ ($D$) for which $C_i \cup X'$ ($X'$) is a clique in $G$.

The formal correctness proof is straightforward and thus omitted. It remains to show the running time. For each $X \subseteq S$ and each $i \in [0, |\mathcal{C}_{V \setminus S}|]$, the recurrence for $T[X, i]$ and $D[X]$ can both be evaluated in $2^{|X|} \cdot n^{\mathcal{O}(1)}$ time. Since for each $j \in [0, |S|]$, there are exactly $\binom{|S|}{j}$ size-$j$ subsets of $S$, all entries of both dynamic programming tables can be evaluated in $\sum_{j=0}^{|S|} \binom{|S|}{j} \cdot 2^j \cdot n^{\mathcal{O}(1)} = 3^{|S|} \cdot n^{\mathcal{O}(1)}$ total time. □

Based on Theorem 2.17, we thus achieve an algorithm that runs in time $(3 \cdot e)^k \cdot \Delta^{2k} \cdot n^{\mathcal{O}(1)}$ for both LS Cluster Deletion and LS Cluster Editing, since for both problems, $G^2$ is a candidate support graph that has a maximum degree of $\Delta^2$.

**Theorem 5.6.** LS Cluster Editing *and* LS Cluster Deletion *can be solved in* $(3 \cdot e)^k \cdot \Delta^{2k} \cdot n^{\mathcal{O}(1)}$ *time.*

We remark that an algorithm solving LS Cluster Editing or LS Cluster Deletion in $f(\Delta) \cdot n^{\mathcal{O}(1)}$ time is unlikely: Setting $k := n$ and applying this algorithm repeatedly until no improvement is found would solve Cluster Editing or Cluster Deletion in $f(\Delta) \cdot n^{\mathcal{O}(1)}$ time; this is unlikely as both problems are NP-hard for constant maximum degree [109].

## 5.3 Running Time Lower Bounds

In this section we present several hardness results for LS Cluster Deletion and LS Cluster Editing.

### 5.3.1 A Lower Bound for LS Cluster Deletion

We start by presenting our hardness result for LS Cluster Deletion that also hold for the permissive version.

**Theorem 5.7.** LS Cluster Deletion *is*

- W[1]*-hard when parameterized by* $k+\ell$*, where* $\ell := \max_{C\in\mathcal{C}} |C|$ *denotes the size of the largest cluster in* $\mathcal{C}$*, and*
- *cannot be solved in* $f(k+\ell)\cdot n^{o(k+\ell)}$ *time for any computable function* $f$*, unless the ETH fails.*

*All of this holds even if the* $k$*-move neighborhood of* $\mathcal{C}$ *contains an optimal proper clustering* $\mathcal{C}^*$*.*

*Proof.* We present a polynomial time reduction from Multicolored Clique. Recall that Multicolored Clique is W[1]-hard when parameterized by the size of the sought clique [35].

Let $I := (G = (V, E), k)$ be an instance of Multicolored Clique and let $V_k$ be the largest part of the $k$-partition $(V_1, \ldots, V_k)$ of $G$. We define an instance $I' := (G' := (V', E'), k', \mathcal{C})$ of LS Cluster Deletion as follows: Initialize $G'$ as a copy of $G$. Then for each $i \in [1, k-1]$, do the following:

- add a vertex $x_i$ to $G'$ and
- for each vertex $v \in V_i$, add a vertex set $K_v$ of size $z := 2k$ to $G'$ and turn $K_v \cup \{v\}$ and $K_v \cup \{x_i\}$ into cliques in $G'$.

Let $X := \{x_i \mid 1 \leq i \leq k-1\}$. Finally, add two additional adjacent vertices $a$ and $b$ to $G'$ and make $X \cup \{a\}$ a clique in $G'$. This completes the construction of $G'$. We complete the construction of $I'$ by setting $k' := 2k-1$ and

$$\mathcal{C} := \{X \cup \{a\}, \{b\}\} \cup \bigcup_{v \in V \setminus V_k} \{K_v \cup \{v\}\} \cup \bigcup_{v \in V_k} \{\{v\}\}.$$

Note that by construction, $\mathcal{C}$ is a proper clustering of $G'$ and $|\mathrm{Cl}(\mathcal{C})| = |V \setminus V_k| \cdot \binom{z+1}{2} + \binom{k}{2}$. For $k = 4$, an example of the initial clustering and an improving clustering (if one exists) is depicted in Figure 5.2.

Next, we show that there is a clique of size $k$ in $G$ if and only if there is a clustering $\mathcal{C}'$ of $G'$ that improves over $\mathcal{C}$. We further show that each clustering $\mathcal{C}'$ of $G'$ that improves over $\mathcal{C}$ is a $k'$-move neighbor of $\mathcal{C}$.

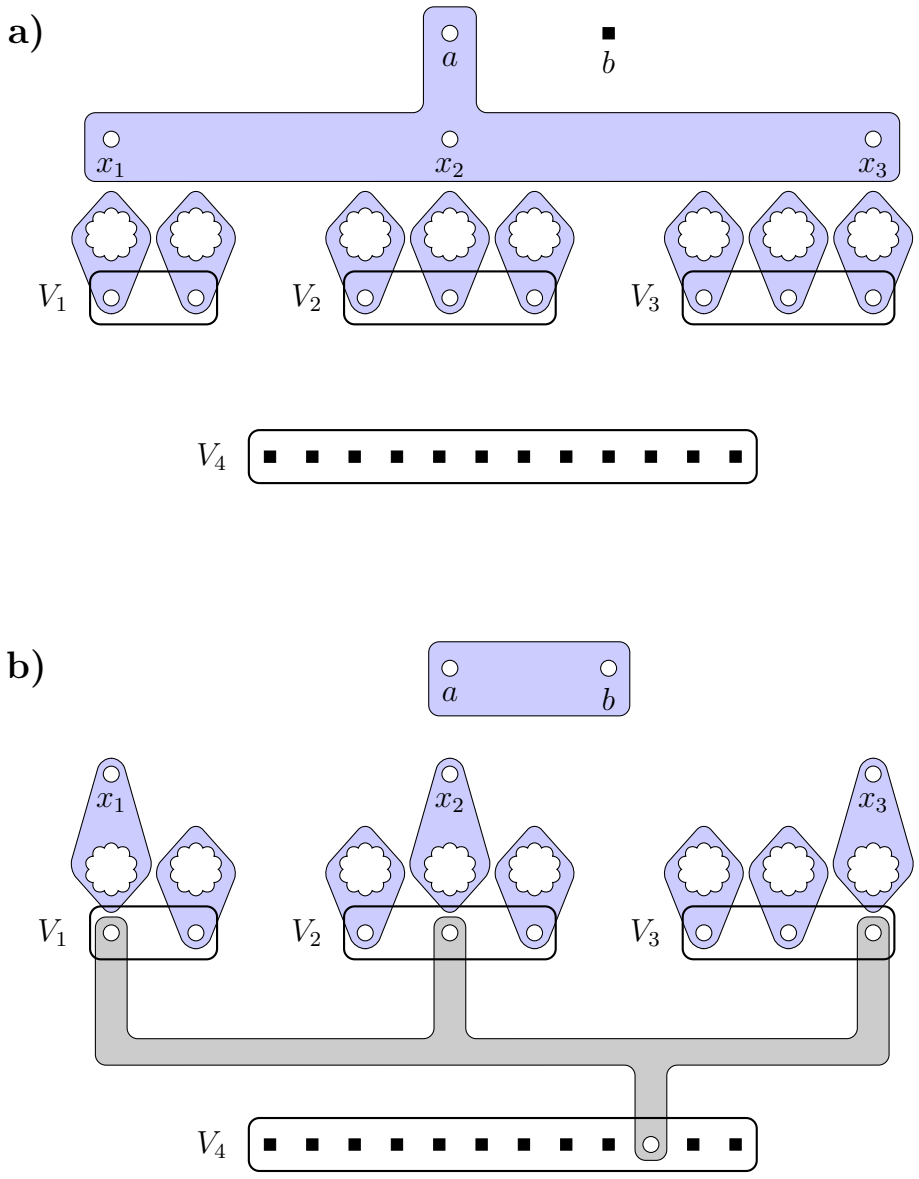



**Figure 5.2:** Two solutions for the instance of LS Cluster Deletion constructed in the proof of Theorem 5.7 for $k = 4$. In both solutions, the clouds indicate the cliques $K_v$ for each vertex $v \in V \setminus V_k$ and squared black vertices represent clusters only containing a single vertex. a) shows the initial solution. b) shows an improving solution, if one exists. Such an improving solution only exists, if there is a clique of size $k$ in the graph of the Multicolored Clique instance. This clique is indicated by the gray cluster in the lower clustering.

($\Rightarrow$) Let $S$ be a clique of size $k$ in $G$. Since $(V_1, \ldots, V_k)$ is a $k$-partition of $G$, for each $i \in [1, k]$, $S$ contains exactly one vertex of $V_i$. For each $i \in [1, k]$, let $v_i$ be that unique vertex of $V_i \cap S$. We set

$$\mathcal{C}' := \{S, \{a, b\}\} \cup \bigcup_{i \in [1,k-1]} \{K_{v_i} \cup \{x_i\}\} \cup \bigcup_{v \in V \setminus (V_k \cup S)} \{K_v \cup \{v\}\} \cup \bigcup_{v \in V_k \setminus \{v_k\}} \{\{v\}\}.$$

Note that by construction, $\mathcal{C}'$ is a proper clustering of $G'$ and $|\mathrm{Cl}(\mathcal{C}')| = \binom{k}{2} + |X| \cdot \binom{z+1}{2} + |V \setminus (V_k \cup S)| \cdot \binom{z+1}{2} + 1 = |V \setminus V_k| \cdot \binom{z+1}{2} + \binom{k}{2} + 1 = |\mathrm{Cl}(\mathcal{C})| + 1$ by definition of $\mathcal{C}'$. Hence, $\mathcal{C}'$ is improving over $\mathcal{C}$. Moreover, $\mathcal{C}'$ and $\mathcal{C}$ are $k'$-move neighbors.

($\Leftarrow$) Let $\mathcal{C}'$ be a best proper clustering of $G'$ and suppose that $\mathcal{C}'$ improves over $\mathcal{C}$. Before we show that there is a clique of size $k$ in $G$, we prove some properties of the clustering $\mathcal{C}'$.

First, we show that for each vertex $v \in V \setminus V_k$, there is a cluster $C'_v$ in $\mathcal{C}'$ with $K_v \subseteq C'_v$. Suppose that there are at least two clusters $C'_v$ and $C''_v$ in $\mathcal{C}'$ with $K_v \cap C'_v \neq \emptyset$ and $K_v \cap C''_v \neq \emptyset$ and suppose that $|C'_v| \geq |C''_v|$ . Let $v'$ be an arbitrary vertex of $C''_v \cap K_v$. Recall that by construction of $G'$, $v'$ has the same closed neighborhood as any other vertex of $K_v$. Since $\mathcal{C}'$ is a proper clustering of $G'$, each vertex of $C'_v$ is part of the closed neighborhood of $v'$. Hence, the clustering $\mathcal{C}'' := (\mathcal{C} \setminus \{C'_v, C''_v\}) \cup \{C'_v \cup \{v'\}, C''_v \setminus \{v'\}\}$ is a proper clustering of $G'$ and improves over $\mathcal{C}'$. Since $\mathcal{C}'$ is a best proper clustering of $G'$, no such two clusters exist and thus for each vertex $v \in V \setminus V_k$, there is a cluster $C'_v$ in $\mathcal{C}'$ with $K_v \subseteq C'_v$.

Next, we show that for each $i \in [1, k-1]$ and each vertex $v \in V_i$, the cluster $C'_v$ is either $K_v \cup \{v\}$ or $K_v \cup \{x_i\}$. Suppose that this is not the case and let $i \in [1, k-1]$ and let $v$ be a vertex of $V_i$ such that $C'_v \notin \{K_v \cup \{v\}, K_v \cup \{x_i\}\}$. Note that this implies that $C'_v = K_v$. Furthermore, let $C''$ be the cluster of $\mathcal{C}'$ that contains $v$. Since $\mathcal{C}'$ is a proper clustering of $G'$ and $v$ has only neighbors in $V \cup K_v$, the cluster $C''$ is a clique in $G$. Moreover, since $G$ is $k$-partite, this implies that $C''$ has size at most $k$. Hence, the clustering $\mathcal{C}'' := (\mathcal{C} \setminus \{C'_v, C''\}) \cup \{K_v \cup \{v\}, C'' \setminus \{v\}\}$ is a proper clustering of $G'$ and improves over $\mathcal{C}'$, since $C'_v$ has size $2k > k$. Since $\mathcal{C}'$ is a best proper clustering of $G'$, this implies that for each $i \in [1, k-1]$ and each vertex $v \in V_i$, the cluster $C'_v$ is either $K_v \cup \{v\}$ or $K_v \cup \{x_i\}$.

Note that this implies that for each $i \in [1, k-1]$, there is at most one vertex $v_i \in V_i$ for which $C'_{v_i} \neq K_v \cup \{v_i\}$. Intuitively, if such a vertex $v_i$ moves out of its cluster from $\mathcal{C}$, then the vertex $x_i$ has to move into the original cluster of $v_i$.

Let $S' \subseteq V'$ be a minimal set of vertices that have to move to obtain $\mathcal{C}'$ from $\mathcal{C}$. Moreover, let $S := S' \cap (V \setminus V_k)$. By the above, for each $i \in [1, k-1]$, $S$ contains at most one vertex of $V_i$. Recall that each vertex of $V_k$ has neighbors only in $V \setminus V_k$ and can thus only be in a cluster of $\mathcal{C}'$ with a (potentially empty) subset of vertices

of $S$. Hence, $S'$ contains no vertex of $V_k$. In the following, we show that there is a vertex $v_k \in V_k$ such that $S \cup \{v_k\}$ is a clique of size $k$ in $G$.

To this end, we analyze the number of edges in the clusters $\mathcal{C}_S \subseteq \mathcal{C}'$ that contain vertices of $S \cup V_k$ and have size at least two, and the number of edges in the clusters $\mathcal{C}_X \subseteq \mathcal{C}'$ that have size at least two and contain only vertices of $X \cup \{a, b\}$. By the above, each cluster $C$ of $\mathcal{C}_S$ contains only vertices of $S \cup V_k$ and $C$ contains at most one vertex of $V_k$, since $V_k$ is an independent set in $G'$. Note that this implies that each cluster of $\mathcal{C}_S$ contains at least one vertex of $S$. Furthermore, note that $\mathrm{Cl}(\mathcal{C}') = \bigcup_{v \in V \setminus V_k} \binom{C'_v}{2} \cup \mathrm{Cl}(\mathcal{C}_S) \cup \mathrm{Cl}(\mathcal{C}_X)$. Since for each vertex $v \in V \setminus V_k$, $C'_v$ has size $z+1$, $|\mathrm{Cl}(\mathcal{C}')| = |V \setminus V_k| \cdot \binom{z+1}{2} + |\mathrm{Cl}(\mathcal{C}_S)| + |\mathrm{Cl}(\mathcal{C}_X)|$. Moreover, since $\mathcal{C}'$ is improving over $\mathcal{C}$, $|\mathrm{Cl}(\mathcal{C}')| - |\mathrm{Cl}(\mathcal{C})| = |\mathrm{Cl}(\mathcal{C}_S)| + |\mathrm{Cl}(\mathcal{C}_X)| - \binom{k}{2} \geq 1$. In the following, we show that $|\mathrm{Cl}(\mathcal{C}_S)| + |\mathrm{Cl}(\mathcal{C}_X)| \leq \binom{k}{2} + 1$ and that $|\mathrm{Cl}(\mathcal{C}_S)| + |\mathrm{Cl}(\mathcal{C}_X)| \leq \binom{k}{2}$ if there is no vertex $v_k \in V_k$ such that $S \cup \{v_k\}$ is a clique of size $k$ in $G$. To this end, we analyze the size of $\mathrm{Cl}(\mathcal{C}_S)$ and the size of $\mathrm{Cl}(\mathcal{C}_X)$ separately.

First, we show that if $\mathcal{C}_S$ has size at least two, then $|\mathrm{Cl}(\mathcal{C}_S)| < \binom{|S|+1}{2}$. This follows inductively by the fact that each cluster of $\mathcal{C}_S$ has size at least two and contains at most one vertex of $V_k$, and for each $i \geq 2$ and each $j \geq 2$, $\binom{i}{2} + \binom{j}{2} < \binom{i+j-1}{2}$. Hence, $\mathrm{Cl}(\mathcal{C}_S)$ has size at least $\binom{|S|+1}{2}$ if and only if there is a vertex $v_k \in V_k$ such that $\mathcal{C}_S = \{S \cup \{v_k\}\}$. Moreover, this implies that $|\mathrm{Cl}(\mathcal{C}_S)| \leq \binom{|S|+1}{2}$.

Second, we bound the size of $\mathrm{Cl}(\mathcal{C}_X)$. Since for each $i \in [1, k-1]$, if there is a vertex $v_i \in V_i \cap S$, then the vertex $x_i$ moves to the cluster containing all vertices of $K_{v_i}$, that is, $x_i$ is not contained in any cluster of $\mathcal{C}_X$. Hence, at most $k - 1 - |S|$ vertices of $X$ are contained in clusters of $\mathcal{C}_X$. Since $b$ is adjacent only to $a$, this implies that $|\mathrm{Cl}(\mathcal{C}_X)| \leq \binom{k-|S|}{2} + 1$.

Finally, we show that for some vertex $v_k \in V_k$, $S \cup \{v_k\}$ is a clique of size $k$ in $G$. Assume that $|S| < k-1$. Hence, by the above $|\mathrm{Cl}(\mathcal{C}_S)| + |\mathrm{Cl}(\mathcal{C}_X)| \leq \binom{|S|+1}{2} + \binom{k-|S|}{2} + 1 < \binom{k}{2} + 1$ and thus $|\mathrm{Cl}(\mathcal{C}_S)| + |\mathrm{Cl}(\mathcal{C}_X)| \leq \binom{k}{2}$. Since $|\mathrm{Cl}(\mathcal{C}')| > |\mathrm{Cl}(\mathcal{C})|$, this is not possible.

Consequently, $|S| = k - 1$. Hence, by the above $|\mathrm{Cl}(\mathcal{C}_S)| + |\mathrm{Cl}(\mathcal{C}_X)| \leq \binom{|S|+1}{2} + \binom{k-|S|}{2} + 1 = \binom{k}{2} + 1$. Hence, $|\mathrm{Cl}(\mathcal{C}')| \leq |\mathrm{Cl}(\mathcal{C})| + 1$. Since $\mathcal{C}'$ improves over $\mathcal{C}$, $|\mathrm{Cl}(\mathcal{C}_S)| + |\mathrm{Cl}(\mathcal{C}_X)| = \binom{k}{2} + 1$. As shown before, $|\mathrm{Cl}(\mathcal{C}_S)| + |\mathrm{Cl}(\mathcal{C}_X)| = \binom{k}{2} + 1$ only if $\mathcal{C}_S$ consists of a single cluster $C' := S \cup \{v_k\}$ for some vertex $v_k \in V_k$. Hence, there is a clique of size $k$ in $G$ and $I$ is a yes-instance of Multicolored Clique.

Moreover, note that this implies that $\mathcal{C}'$ is a $k'$-move neighbor of $\mathcal{C}$, since only the $k-1$ vertices of $S$, the $k-1$ vertices of $X$, and either $a$ or $b$ changed their cluster.

This completes the correctness of the reduction. Next we discuss the implications for lower bounds. Let $\ell := \max_{C \in \mathcal{C}} |C|$. Recall that $\ell = 2k + 1$ since $z = 2k$.

Since Multicolored Clique is W[1]-hard when parameterized by $k$ and cannot be solved in $f(k)\cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails [35], this implies that LS Cluster Deletion is W[1]-hard when parameterized by $k'+\ell$ and cannot be solved in $f(k'+\ell)\cdot|V'|^{o(k'+\ell)}$ time for any computable function $f$, unless the ETH fails, since $|V'|\in n^{\mathcal{O}(k)}$. □

Note that due to the last property of Theorem 5.7, the permissive version of LS Cluster Deletion shares the same ETH-based lower bound and does not admit an FPT-algorithm with respect to $k+\ell$, unless FPT = W[1].

### 5.3.2 Auxiliary Lower Bounds for Densest-$k$-Subgraph

In the remainder of the section, we present our hardness results for LS Cluster Editing. These are obtained by reductions from restricted instances of Densest-$k$-Subgraph, which is defined as follows:

Densest-$k$-Subgraph
**Input:** A undirected graph $G=(V,E)$, $k\in\mathbb{N}$, and $d\in\mathbb{N}$.
**Question:** Is there a size-$k$ subset $S\subseteq V$ such that $|E(S)|\geq\binom{k}{2}-d$?

Hence, we first show that Densest-$k$-Subgraph provides these hardness results on the desired restricted instances. In the following, we denote by $\delta(G)$ the degeneracy of $G$.

**Theorem 5.8.** *Even if $d=\frac{k-1}{2}$, Densest-$k$-Subgraph is W[1]-hard when parameterized by $k+\delta(G)$ and cannot be solved in $f(k)\cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. This holds even on instances where $|E(S)|<\binom{k-1}{2}-\frac{k-1}{4}$ for each vertex set $S$ of size $k-1$.*

The proof of Theorem 5.8 is based on three different reductions that use the following fact.

**Fact 5.9.** $\binom{a+b}{2}=\binom{a}{2}+a\cdot b+\binom{b}{2}$.

We now present our first step to prove Theorem 5.8, namely the W[1]-hardness of Densest-$k$-Subgraph when parameterized by $k$ plus the degeneracy of $G$.

**Lemma 5.10.** *Even if $d=k$, Densest-$k$-Subgraph is W[1]-hard when parameterized by $k+\delta(G)$. This holds even on instances where $|E(S)|<\binom{k-1}{2}-\frac{k}{2}$ for each vertex set $S$ of size $k-1$.*

*Proof.* We present a parameterized reduction from Multicolored Clique.

Let $I = (G = (V, E), k)$ be an instance of Multicolored Clique with $k > 4$. Moreover let $G' = (V', E')$ be the graph obtained by subdividing each edge of $G$. Note that $G'$ is a 2-degenerate bipartite graph. Now, $I$ is a yes-instance of Multicolored Clique if and only if $I' := (G', k' := \binom{k}{2} + k, d' := \binom{k'}{2} - 2 \cdot \binom{k}{2})$ is a yes-instance of Densest-$k$-Subgraph. We define an instance $I'' = (G'' = (V'', E''), k'', d'')$ of Densest-$k$-Subgraph as follows: We initialize $G''$ as $G'$ and add a clique $K^*$ of size $k^* := \binom{k'}{2} - 2 \cdot \binom{k}{2} - k'$ to $G''$, such that each vertex of $V$ is adjacent to each vertex of $K^*$. Finally, we set $k'' := k' + k^*$ and $d'' := k''$.

We next show that $I'$ is a yes-instance of Densest-$k$-Subgraph if and only if $I''$ is a yes-instance of Densest-$k$-Subgraph.

($\Rightarrow$) Let $S' \subseteq V'$ be a vertex set of size $k'$ such that $|E_{G'}(S')| \geq 2 \cdot \binom{k}{2}$. We set $S'' := S' \cup K^*$ and show that $|E_{G''}(S'')| \geq \binom{k''}{2} - k''$. Since each vertex of $K^*$ is adjacent to each other vertex in $S''$, there are $\binom{k^*}{2} + k^* \cdot k'$ edges incident with vertices of $K^*$ in $S''$. Additionally, $E_{G''}(S'')$ contains exactly the edges of $E_{G'}(S')$. Hence,

$$\begin{aligned}
|E_{G''}(S'')| &= \binom{k^*}{2} + k^* \cdot k' + |E_{G'}(S')| \\
&\geq \binom{k^*}{2} + k^* \cdot k' + \binom{k'}{2} - \left(\binom{k'}{2} - 2 \cdot \binom{k}{2}\right) \\
&\overset{\text{Fact 5.9}}{=} \binom{k^* + k'}{2} - \left(\binom{k'}{2} - 2 \cdot \binom{k}{2} - k' + k'\right) \\
&= \binom{k''}{2} - (k^* + k') = \binom{k''}{2} - d''.
\end{aligned}$$

Hence, $I''$ is a yes-instance of Densest-$k$-Subgraph.

($\Leftarrow$) Let $S'' \subseteq V''$ be a vertex set of size $k''$ such that $|E_{G''}(S'')| \geq \binom{k''}{2} - d''$. We can assume without loss of generality that $S''$ contains all vertices of $K^*$ since each of these vertices is adjacent to each other vertex of $G''$. We set $S' := S'' \setminus K^*$ and show that $|E_{G'}(S')| \geq 2 \cdot \binom{k}{2}$. Since each vertex of $K^*$ is adjacent to each other vertex in $S''$, there are $\binom{k^*}{2} + k^* \cdot k'$ edges incident with vertices of $K^*$ in $S''$. Additionally,

$E_{G''}(S'')$ contains exactly the edges of $E_{G'}(S')$. Hence,

$$\begin{aligned}|E_{G'}(S')| &= |E_{G''}(S'')| - \left(\binom{k^*}{2} + k^* \cdot k'\right) \geq \binom{k''}{2} - d'' - \left(\binom{k^*}{2} + k^* \cdot k'\right) \\ &\overset{\text{Fact 5.9}}{=} \binom{k^*}{2} + k^* \cdot k' + \binom{k'}{2} - d'' - \left(\binom{k^*}{2} + k^* \cdot k'\right) = \binom{k'}{2} - d'' \\ &= \binom{k'}{2} - (k^* + k') = \binom{k'}{2} - \left(\binom{k'}{2} - 2 \cdot \binom{k}{2} - k' + k'\right) = 2 \cdot \binom{k}{2}.\end{aligned}$$

Hence, $I'$ is a yes-instance of Densest-$k$-Subgraph.

**Instance restrictions.** By construction, $G'$ is 2-degenerate. Since $G''$ consists of $G'$ and $k^* \in \mathcal{O}(k^4)$ additional vertices, it follows that the degeneracy of $G''$ is $\mathcal{O}(k^4)$.

It remains to show that $|E_{G''}(S'')| < \binom{k''-1}{2} - \frac{k''}{2}$ for each vertex set $S'' \subseteq V''$ of size $k-1$. Let $S'' \subseteq V''$ be any vertex set of size $k''-1$ that maximizes $|E_{G''}(S'')|$. Hence, we can assume that $S''$ contains all vertices of $K^*$. Let $S' := S'' \setminus K^*$. Hence, $S'$ has size $k'-1$. Since $G'$ is 2-degenerate, $|E_{G'}(S')| \leq 2 \cdot (k'-1) < k^2 + k$.

Hence, $|E_{G''}(S'')| \leq \binom{k^*}{2} + k^* \cdot (k'-1) + |E_{G'}(S')| < \binom{k^*}{2} + k^* \cdot (k'-1) + k^2 + k$. We show that this is smaller than $\binom{k''-1}{2} - \frac{k''}{2}$. Since

$$\begin{aligned}\binom{k''-1}{2} - \frac{k''}{2} &= \binom{k^* + (k'-1)}{2} - \frac{k^* + k'}{2} \\ &= \binom{k^*}{2} + k^* \cdot (k'-1) + \binom{k'-1}{2} - \frac{\binom{k'}{2} - 2 \cdot \binom{k}{2}}{2},\end{aligned}$$

it remains to show that $k^2 + k < \binom{k'-1}{2} - \frac{\binom{k'}{2} - 2 \cdot \binom{k}{2}}{2}$. This is the case if and only if

$$\begin{aligned}0 &< 2 \cdot \binom{k'-1}{2} - \binom{k'}{2} + 2 \cdot \binom{k}{2} - 2k^2 - 2k \\ &= \binom{k'-1}{2} + \left(\binom{k'-1}{2} - \binom{k'}{2}\right) + 2 \cdot \binom{k}{2} - 2k^2 - 2k \\ &= \binom{k'-1}{2} - (k'-1) + 2 \cdot \binom{k}{2} - 2k^2 - 2k \\ &= \binom{\binom{k+1}{2} - 1}{2} - \left(\binom{k+1}{2} - 1 - 2 \cdot \binom{k}{2} + 2k^2 + 2k\right).\end{aligned}$$

This inequality holds, since $k > 4$. Consequently, $|E_{G''}(S'')| < \binom{k''-1}{2} - \frac{k''}{2}$ for each vertex set $S'' \subseteq V''$ of size $k-1$, which proves the statement. □

Next, we present our second step to prove Theorem 5.8, namely an extension of the previous theorem to cases where $d = \frac{k-1}{2}$.

**Lemma 5.11.** *Even if $d = \frac{k-1}{2}$,* Densest-$k$-Subgraph *is* W[1]*-hard when parameterized by $k+\delta(G)$. This holds even on instances where $k$ is odd and where $|E(S)| < \binom{k-1}{2} - \frac{k-1}{4}$ for each vertex set $S$ of size $k-1$.*

*Proof.* We present a parameterized reduction from Densest-$k$-Subgraph. Let $I := (G := (V, E), k, d)$ be an instance of Densest-$k$-Subgraph with the restrictions described in Lemma 5.10, that is, $d = k$ and $|E(S)| < \binom{k-1}{2} - \frac{k}{2}$ for each vertex set $S \subseteq V$ of size $k-1$. We define an instance $I' := (G' = (V', E'), k', d')$ of Densest-$k$-Subgraph where $d' = \frac{k'-1}{2}$ as follows: We obtain $G'$ by adding a clique $K^*$ of size $k^* := k + 1$ to $G$ such that each vertex of $V$ is adjacent to each vertex of $K^*$. Finally, we set $k' := k + k^* = 2k + 1$ and $d' := \frac{k'-1}{2} = d$. Next, we show that $I$ is a yes-instance of Densest-$k$-Subgraph if and only if $I'$ is a yes-instance of Densest-$k$-Subgraph.

($\Rightarrow$) Let $S$ be a vertex set of size $k$ in $G$ such that $|E_G(S)| \geq \binom{k}{2} - d$. We set $S' := S \cup K^*$ and show that $|E_{G'}(S')| \geq \binom{k'}{2} - d' = \binom{k'}{2} - d$. Since each vertex of $K^*$ is adjacent to each other vertex of $S'$, $|E_{G'}(S')| = \binom{k^*}{2} + k^* \cdot k + |E_G(S)| \geq \binom{k^*}{2} + k^* \cdot k + \binom{k}{2} - d = \binom{k^*+k}{2} - d = \binom{k'}{2} - d'$. Hence, $I'$ is a yes-instance of Densest-$k$-Subgraph.

($\Leftarrow$) Let $S'$ be a vertex set of size $k'$ in $G'$ such that $|E_{G'}(S')| \geq \binom{k'}{2} - d'$. We can assume that $S'$ contains all vertices of $K^*$, since each vertex of $K^*$ is adjacent to each other vertex of $V'$. We set $S := S' \setminus K^*$ and show that $|E_G(S)| \geq \binom{k}{2} - d = \binom{k}{2} - d'$. Since each vertex of $K^*$ is adjacent to each other vertex of $S'$, $|E_G(S)| = |E_{G'}(S')| - \binom{k^*}{2} - k^* \cdot k \geq \binom{k'}{2} - d' - \binom{k^*}{2} - k^* \cdot k = \binom{k+k^*}{2} - d' - \binom{k^*}{2} - k^* \cdot k = \binom{k^*}{2} + k^* \cdot k + \binom{k}{2} - d' - \binom{k^*}{2} - k^* \cdot k = \binom{k}{2} - d' = \binom{k}{2} - d$. Hence, $I$ is a yes-instance of Densest-$k$-Subgraph.

**Parameter bounds and instance restrictions.** Recall that Densest-$k$-Subgraph is W[1]-hard when parameterized by $k+\delta(G)$ even if $d = k$ (Lemma 5.10). Since we only added $k + 1$ additional vertices to $G$ and no edges between vertices of $V$ to obtain $G'$, $\delta(G') \leq \delta(G)+k+1$. Hence, Densest-$k$-Subgraph is W[1]-hard when parameterized by $k' + \delta(G')$ even if $d' = \frac{k'-1}{2}$.

Finally, we show that $|E_{G'}(S')| < \binom{k'-1}{2} - \frac{k'-1}{4}$ for each vertex set $S' \subseteq V'$ of size $k' - 1$. Let $S' \subseteq V'$ be any vertex set of size $k' - 1$ that maximizes $|E_{G'}(S')|$. Hence, we can assume that $S'$ contains all vertices of $K^*$. Let $S := S' \setminus K^*$. Hence, $S$ has size $k-1$. Thus, $|E_{G'}(S')| = \binom{k^*}{2} + k^* \cdot (k-1) + |E_G(S)|$. We show that $|E_{G'}(S')| < \binom{k'-1}{2} - \frac{k'-1}{4} = \binom{k'-1}{2} - \frac{k}{2}$. By assumption, $|E_G(S)| < \binom{k-1}{2} - \frac{k}{2}$. Hence, it remains to

show that $\binom{k^*}{2} + k^* \cdot (k-1) + \binom{k-1}{2} - \frac{k}{2} \leq \binom{k'-1}{2} - \frac{k}{2}$. Since $\binom{k^*}{2} + k^* \cdot (k-1) + \binom{k-1}{2} = \binom{k^*+(k-1)}{2} = \binom{k'-1}{2}$, the inequality holds. □

To prove Theorem 5.8, it thus remains to show that Densest-$k$-Subgraph has the desired ETH-based running time lower bound on these restricted instances. To prove this, we make use of the following fact.

**Observation 5.12.** *Let $x \in \mathbb{N}$, let $G = (V, E)$ be an $x$-partite graph with $x$-partition $(V_1, \ldots, V_x)$, and let $S$ be a vertex set of size $y$ with $x \leq y \leq 2x$. Then, $|E(S)| \leq \binom{y}{2} - (y - x)$. Moreover, if $|E(S)| = \binom{y}{2} - (y - x)$, then for each $i \in [1, x]$, $|V_i \cap S| \in \{1, 2\}$ and for each vertex $v_i \in V_i \cap S$, $v_i$ is adjacent to each vertex of $S \setminus V_i$.*

We are now ready to present the third and final step to prove Theorem 5.8.

**Lemma 5.13.** Densest-$k$-Subgraph *cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. This holds even if $d = \frac{k-1}{2}$ on instances where $k$ is odd and where $|E(S)| < \binom{k-1}{2} - \frac{k-1}{4}$ for each vertex set $S$ of size $k - 1$.*

*Proof.* We present a polynomial time reduction from Multicolored Clique [35]. Let $I := (G := (V, E), k)$ be an instance of Multicolored Clique with $k > 1$ and let $(V_1, \ldots, V_k)$ be the $k$-partition of $G$. We define an instance $I' := (G' = (V', E'), k', d')$ of Densest-$k$-Subgraph where $d' = \frac{k'-1}{2}$ as follows: We obtain $G'$ by adding a clique $K^* := \{v_i^* \mid 1 \leq i \leq k\}$ to $G$ such that each vertex $v_i^*$ is adjacent to each vertex of $V \setminus V_i$. Additionally, we add a vertex $v^*$ to $G'$ which is adjacent to each other vertex. Finally, we set $k' := 2k + 1$ and $d' := \frac{k'-1}{2} = k$. Note that $G'$ is $(k+1)$-partite with $(k+1)$-partition $(V_1', \ldots, V_{k+1}')$, where $V_{k+1}' := \{v^*\}$ and $V_i' := V_i \cup \{v_i^*\}$ for each $i \in [1, k]$. Next, we show that $I$ is a yes-instance of Multicolored Clique if and only if $I'$ is a yes-instance of Densest-$k$-Subgraph.

($\Rightarrow$) Let $S$ be a clique of size $k$ in $G$. We set $S' := S \cup K^* \cup \{v^*\}$ and show that $|E_{G'}(S')| \geq \binom{k'}{2} - d' = \binom{k'}{2} - k$. Since $S$ is a clique of size $k$ in $G$ and $G$ is $k$-partite, $S$ contains for each $i \in [1, k]$ exactly one vertex $v_i$ of $V_i$. Moreover, since $v^*$ is adjacent to each other vertex in $S'$ and since for each $i \in [1, k]$, $v_i^*$ is adjacent to each other vertex of $S'$ besides $v_i$, $|E_{G'}(S')| = |E_{G'}(K^* \cup \{v^*\})| + |E_{G'}(K^* \cup \{v^*\}, S)| + |E_G(S)| = \binom{k+1}{2} + k \cdot (k+1) - k + |E_G(S)|$. Since $S$ is a clique of size $k$ in $G$, $|E_{G'}(S')| = \binom{k+1}{2} + k \cdot (k+1) - k + \binom{k}{2} = \binom{(k+1)+k}{2} - k = \binom{k'}{2} - d'$. Hence, $I'$ is a yes-instance of Densest-$k$-Subgraph.

($\Leftarrow$) Let $S'$ be a vertex set of size $k'$ in $G'$ such that $|E_{G'}(S')| \geq \binom{k'}{2} - d' = \binom{2k+1}{2} - k$. Since $G'$ is $(k+1)$-partite and $S'$ has size $2k+1$, due to Observation 5.12, $|E_{G'}(S')| \leq \binom{2k+1}{2} - k$. Hence, $|E_{G'}(S')| = \binom{2k+1}{2} - k$ and thus, due to Observation 5.12, for each $i \in [1, k+1]$, $|S \cap V_i'| \in \{1, 2\}$ and each vertex of $V_i' \cap S'$ is

adjacent to each vertex of $S' \setminus V'_i$. Since $S'$ has size $2k+1$ and $V'_{k+1} = \{v^*\}$, $S'$ contains for each $i \in [1,k]$ at least one vertex $v_i$ of $V_i$. Let $S := \{v_i \mid 1 \leq i \leq k\}$. Since $S$ is a subset of $S'$, by the above, $S$ is a clique in $G'$ and thus also a clique in $G$. Hence, $I$ is a yes-instance of Multicolored Clique.

**Parameter bounds and instance restrictions.** Recall that Multicolored Clique cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails [35]. Since $k' \in \mathcal{O}(k)$ and $|V'| \in n^{\mathcal{O}(1)}$, this implies that Densest-$k$-Subgraph cannot be solved in $f(k) \cdot |V'|^{o(k)}$ time for any computable function $f$, unless the ETH fails.

Finally, we show that $|E_{G'}(S')| < \binom{k'-1}{2} - \frac{k'-1}{4}$ for each vertex set $S' \subseteq V'$ of size $k'-1$. Recall that $G'$ is $(k+1)$-partite, $S'$ has size $k'-1 = 2k$, and $k > 1$. Hence, due to Observation 5.12, $|E_{G'}(S')| \leq \binom{2k}{2} - (k-1) < \binom{2k}{2} - \frac{k}{2} = \binom{k'}{2} - \frac{k'-1}{4}$. □

Now, Theorem 5.8 follows from Lemmas 5.11 and 5.13.

### 5.3.3 Lower Bounds for LS Cluster Editing

Based on these hardness results for Densest-$k$-Subgraph, we are now able to analyze the parameterized complexity of LS Cluster Editing for the parameter combination of $k$ plus the size of the largest cluster of the initial clustering $\mathcal{C}$.

**Theorem 5.14.** LS Cluster Editing *is* W[1]*-hard when parameterized by* $k + \ell + \delta(G)$*, where* $\ell := \max_{C \in \mathcal{C}} |C|$*. Moreover, unless the ETH fails, there is no computable function* $f$ *for which* LS Cluster Editing *can be solved in* $f(k+\ell) \cdot n^{o(k+\ell)}$ *time.*

*Proof.* We present a parameterized reduction from Densest-$k$-Subgraph with the restrictions listed in Theorem 5.8. Let $I = (G = (V, E), k, d)$ be an instance of Densest-$k$-Subgraph, where $k$ is odd and $d = \frac{k-1}{2}$ such that $|E(S)| < \binom{k-1}{2} - \frac{k-1}{4}$ for each vertex set $S$ of size $k-1$. We define an instance $I' := (G' := (V', E'), k, \mathcal{C})$ of LS Cluster Editing with $\max_{C \in \mathcal{C}} |C| \in \mathcal{O}(k)$ as follows: We initialize $G'$ as $G$ and add for each vertex $v \in V$ a set $K_v$ of $7k + \frac{k-3}{2}$ vertices to $G'$ such that $\{v\} \cup K_v$ is a clique in $G'$. Additionally, we add a clique $K^*$ of size $7k$ to $G'$ and add edges to $G'$, such that each vertex of $V$ is adjacent to each vertex of $K^*$. Finally, we set $\mathcal{C} := \{K^*\} \cup \{\{v\} \cup K_v \mid v \in V\}$. Note that by definition of $\mathcal{C}$, each cluster $C \in \mathcal{C}$ is a clique in $G'$, that is, $\mathcal{C}$ is a proper clustering of $G'$. The correctness proof is based on the following claim.

**Claim 6.** Let $\mathcal{C}' := \{K^* \cup S\} \cup \{\{K_v\} \mid v \in S\} \cup \{\{v\} \cup K_v \mid v \in V \setminus S\}$ be a clustering of $G'$ for some vertex set $S \subseteq V$. The improvement of $\mathcal{C}'$ over $\mathcal{C}$ is $2 \cdot |E_G(S)| - \binom{|S|}{2} - |S| \cdot \frac{k-3}{2}$.

*Proof of Claim.* We only have to consider the edges incident with at least one vertex of $S$ in $\mathrm{Cl}(\mathcal{C})$ and $\mathrm{Cl}(\mathcal{C}')$. Let $F := \mathrm{Cl}(\mathcal{C})$ and let $F' := \mathrm{Cl}(\mathcal{C}')$. Note that the symmetric difference between $F$ and $F'$ are the edges of $F \oplus F' = \{\{v,x\} \mid v \in S, x \in K_v \cup K^*\} \cup \binom{S}{2}$. More precisely, $F \setminus F' = \{\{v,x\} \mid v \in S, x \in K_v\}$ and $F' \setminus F = \{\{v,x\} \mid v \in S, x \in K^*\} \cup \binom{S}{2}$. By construction, all edges of $F \oplus F'$ exist in $G'$, except for the edges of $\binom{S}{2} \setminus E_{G'}(S) = \binom{S}{2} \setminus E_G(S)$. Hence, the improvement of $\mathcal{C}'$ over $\mathcal{C}$ is $\sum_{v \in S}(|K^*| - |K_v|) + |E_{G'}(S)| - \left(\binom{|S|}{2} - |E_{G'}(S)|\right) = 2 \cdot |E_G(S)| - \binom{|S|}{2} - |S| \cdot \frac{k-3}{2}$. This completes the proof. ■

Next, we show that $I$ is a yes-instance of Densest-$k$-Subgraph if and only if $I'$ is a yes-instance of LS Cluster Editing.

($\Rightarrow$) Let $S$ be a set of size $k$ in $G$ such that $|E_G(S)| \geq \binom{k}{2} - d$. We set $\mathcal{C}' := \{K^* \cup S\} \cup \{\{K_v\} \mid v \in S\} \cup \{\{v\} \cup K_v \mid v \in V \setminus S\}$. Note that $d_{\mathrm{move}}(\mathcal{C}, \mathcal{C}') = k$. We show that $\mathcal{C}'$ is improving over $\mathcal{C}$. Due to Claim 6 and since $|E_G(S)| \geq \binom{k}{2} - d$ and $d = \frac{k-1}{2}$, the improvement of $\mathcal{C}'$ over $\mathcal{C}$ is at least

$$\binom{k}{2} - 2d - k \cdot \frac{k-3}{2} = \binom{k}{2} - k + 1 - k \cdot \frac{k-3}{2} = \binom{k}{2} - k \cdot \frac{k-1}{2} + 1 = 1.$$

Hence, $I'$ is a yes-instance of LS Cluster Editing.

($\Leftarrow$) Suppose that $I'$ is a yes-instance of LS Cluster Editing. Let $\mathcal{C}'$ be the best clustering of $G'$ with $d_{\mathrm{move}}(\mathcal{C}, \mathcal{C}') \leq k$. Since $I'$ is a yes-instance of LS Cluster Editing, $\mathcal{C}' \neq \mathcal{C}$. We make some observations about the potential moves between $\mathcal{C}$ and $\mathcal{C}'$. The goal is to show that there is a set $S \subseteq V$ of size at most $k$ such that $\mathcal{C}' = \{K^* \cup S\} \cup \{K_v \mid v \in S\} \cup \{\{v\} \cup K_v \mid v \in V \setminus S\}$. To this end, we show some intermediate results.

First, we show that for each vertex $v \in V$ there is a cluster $C \in \mathcal{C}'$ with $K_v \subseteq C$. Suppose that this is not the case. Hence, there is a vertex $v \in V$ and at least two clusters $C_1$ and $C_2$ in $\mathcal{C}'$ containing vertices of $K_v$. Since $K_v$ has size more than $k$, at least one vertex of $K_v$ is not moved. Assume without loss of generality that $C_1$ contains this vertex. Hence, each vertex of $C_2 \cap K_v$ moved to $C_2$. Thus, $C_2 \cap K_v$ contains at most $k$ vertices. Let $x$ be an arbitrary vertex of $C_2 \cap K_v$. Since $K_v$ has size more than $4k$, $C_1$ contains at least $3k$ vertices of $K_v$ and at most $k$ vertices of $V' \setminus K_v$. By definition of $K_v$, the closed neighborhood of $x$ is exactly $K_v \cup \{v\}$. Hence, $x$ has at most $k$ neighbors in $C_2$, at least $3k$ neighbors in $C_1$ and at most $k$ non-neighbors in $C_1$. Consequently, not moving $x$ to $C_2$ yields a better clustering. Since $\mathcal{C}'$ is the best clustering with $d_{\mathrm{move}}(\mathcal{C}, \mathcal{C}') \leq k$, this is not possible.

Next, we show that there is a cluster $C$ in $\mathcal{C}'$ with $K^* \subseteq C$. Suppose instead that there are at least two clusters $C_1$ and $C_2$ in $\mathcal{C}'$ containing vertices of $K^*$. Since $K^*$

has size more than $k$, at least one vertex of $K^*$ is not moved. Assume without loss of generality that $C_1$ contains this vertex. Hence, each vertex of $C_2 \cap K^*$ moved to $C_2$. Thus, $C_2 \cap K^*$ contains at most $k$ vertices. Let $x$ be an arbitrary vertex of $C_2 \cap K^*$. Since $K^*$ has size more than $4k$, $C_1$ contains at least $3k$ vertices of $K^*$ and at most $k$ vertices of $V' \setminus K^*$. By definition of $K^*$, the closed neighborhood of $x$ is exactly $K^* \cup V$. Hence, since each cluster in $\mathcal{C}$ contains at most one vertex of $V$, $x$ has at most $k+1$ neighbors in $C_2$, at least $3k$ neighbors in $C_1$ and at most $k$ non-neighbors in $C_1$. Consequently, not moving $x$ to $C_2$ yields a better clustering. Since $\mathcal{C}'$ is the best clustering with $d_{\text{move}}(\mathcal{C}, \mathcal{C}') \leq k$, this is not possible.

The above two paragraphs imply that only vertices of $V$ moved to obtain $\mathcal{C}'$ from $\mathcal{C}$. Next, we show that for each vertex $v \in V$, the cluster $C$ of $\mathcal{C}'$ that contains $v$ either contains all vertices of $K^*$ or all vertices of $K_v$. Suppose that this is not the case and let $v$ be a vertex of $V$ such that the cluster $C \in \mathcal{C}'$ with $v \in C$ is not a superset of $K^*$ and not a superset of $K_v$. Since $v$ is only adjacent to at most one vertex in each cluster of $\mathcal{C} \setminus \{K^*, K_v \cup \{v\}\}$, $v$ has at most $k$ neighbors in $C$. Let $K'_v$ be the cluster of $\mathcal{C}'$ containing all vertices of $K_v$. Since $K_v$ has size more than $3k$, $K'_v$ contains at most $k$ non-neighbors of $v$ and at least $3k$ neighbors of $v$. Hence, not moving $v$ from $K'_v$ to $C$ yields a better clustering. Since $\mathcal{C}'$ is the best clustering with $d_{\text{move}}(\mathcal{C}, \mathcal{C}') \leq k$, this is not possible.

Summarizing, there is a nonempty vertex set $S$ of size at most $k$ such that $\mathcal{C}' = \{K^* \cup S\} \cup \{K_v \mid v \in S\} \cup \{\{v\} \cup K_v \mid v \in V \setminus S\}$. More precisely, the vertices of $S$ are exactly the vertices that are moved to obtain $\mathcal{C}'$ from $\mathcal{C}$.

It remains to show that $S$ has size $k$ and that $E_G(S) = E_{G'}(S)$ contains at least $\binom{k}{2} - d$ edges, that is, $|E_G(S)| \geq \binom{k}{2} - d$.

**Claim 7.** $S$ has size $k$.

*Proof of Claim.* Due to Claim 6 and since $\mathcal{C}'$ is improving over $\mathcal{C}$, $2 \cdot |E_G(S)| \geq |S| \cdot \frac{k-3}{2} + \binom{|S|}{2} + 1$.

If $|S| < k-1$, then $2 \cdot |E_G(S)| \leq 2 \cdot \binom{|S|}{2} < |S| \cdot \frac{k-3}{2} + \binom{|S|}{2} + 1$. Since $2 \cdot |E_G(S)| \geq |S| \cdot \frac{k-3}{2} + \binom{|S|}{2} + 1$, we conclude $|S| \geq k-1$.

Assume towards a contradiction that $S$ has size exactly $k-1$. By assumption, $|E_G(S)| < \binom{k-1}{2} - \frac{k-1}{4} = (k-1) \cdot (\frac{k-2}{2} - \frac{1}{4}) = (k-1) \cdot \frac{2k-5}{4}$. Hence, $(k-1) \cdot \frac{2k-5}{2} > 2 \cdot |E_G(S)| \geq |S| \cdot \frac{k-3}{2} + \binom{|S|}{2} + 1 = (k-1) \cdot \frac{k-3}{2} + \binom{(k-1)}{2} + 1 = (k-1) \cdot \frac{2k-5}{2} + 1$, a contradiction.

Consequently, $S$ contains exactly $k$ vertices since to obtain $\mathcal{C}'$ from $\mathcal{C}$, each vertex of $S$ moved and $d_{\text{move}}(\mathcal{C}, \mathcal{C}') \leq k$. ■

Finally, we show that $|E_G(S)| \geq \binom{k}{2} - d$. By Claim 6, $2 \cdot |E_G(S)| \geq |S| \cdot \frac{k-3}{2} + \binom{|S|}{2} + 1 = \frac{2k^2-4k}{2} + 1 = k^2 - k - (k-1)$. Thus $|E_G(S)| \geq \frac{k^2-k}{2} - \frac{k-1}{2} = \binom{k}{2} - d$.

Hence, $I$ is a yes-instance of Densest-$k$-Subgraph.

**Parameter bounds.** Recall that the size $\ell := \max_{C \in \mathcal{C}} |C|$ of the largest cluster in $\mathcal{C}$ is $\mathcal{O}(k)$.

Due to Lemma 5.13, Densest-$k$-Subgraph cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. This implies that LS Cluster Deletion cannot be solved in $f(k+\ell) \cdot |V'|^{o(k+\ell)}$ time for any computable function $f$, unless the ETH fails, since $|V'| \in n^{\mathcal{O}(1)}$.

Next, we analyze the degeneracy of $G'$. Since each vertex of $K_v$ for some vertex $v \in V$ has only neighbors in $K_v \cup \{v\}$, each such vertex has degree $\mathcal{O}(k)$ in $G'$. Furthermore, each vertex of $V$ has only $|K_v| + |K^*| \in \mathcal{O}(k)$ additional neighbors in $G'$. Hence, $\delta(G') \in \mathcal{O}(k + \delta(G))$. Consequently, LS Cluster Editing is W[1]-hard when parameterized by $k + \ell + \delta(G')$, since due to Theorem 5.8, Densest-$k$-Subgraph is W[1]-hard when parameterized by $k + \delta(G)$. □

Next, we show that even when the initial clustering consists only of two clusters, LS Cluster Editing remains W[1]-hard when parameterized by $k$.

**Theorem 5.15.** *Even when the initial clustering consists of only two clusters,* LS Cluster Editing *is* W[1]*-hard when parameterized by $k$ and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails.*

The prove Theorem 5.15, we first show the following auxiliary result.

**Lemma 5.16.** *Even if $d = \frac{k-1}{4}$,* Densest-$k$-Subgraph *is* W[1]*-hard when parameterized by $k$ and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. This holds even if there is an integer $r$ such that each vertex $v \in V$ has degree either 1 or $r$.*

*Proof.* We show this statement in two steps. In the first step, we reduce from Multicolored Clique to Densest-$k$-Subgraph where $d = \frac{k-1}{4}$. Afterwards, we construct an equivalent instance where each vertex has degree either 1 or $r$.

The first reduction is similar to the one in the proof of Lemma 5.13. Let $I := (G := (V, E), k)$ be an instance of Multicolored Clique where $(V_1, \ldots, V_k)$ is the $k$-partition of $G$. We define an instance $I' := (G' = (V', E'), k', d')$ of Densest-$k$-Subgraph where $d' = \frac{k'-1}{4}$ as follows: We obtain $G'$ by adding a clique $K^* := \{v_i^* \mid 1 \leq i \leq k\}$ to $G$ such that for each $i \in [1, k]$, $v_i^*$ is adjacent to each vertex of $V \setminus V_i$. Additionally, we add a clique $\widetilde{K}$ of size $2k+1$ to $G'$ and add edges such that each vertex of $G'$ is adjacent to each vertex of $\widetilde{K}$. Finally, we set $k' := 4k+1$ and $d' := \frac{k'-1}{4} = k$. Note that $G'$ is $(3k+1)$-partite with $(3k+1)$-partition $(V_1', \ldots, V_{3k+1}')$, where for each $i \in [1, k]$, $V_i' := V_i \cup \{v_i^*\}$ and where for each $i \in [k+1, 3k+1]$, $V_i'$ consists

of a single vertex of $\widetilde{K}$. Next, we show that $I$ is a yes-instance of Multicolored Clique if and only if $I'$ is a yes-instance of Densest-$k$-Subgraph.

($\Rightarrow$) Let $S$ be a clique of size $k$ in $G$. We set $S' := S \cup K^* \cup \widetilde{K}$ and show that $|E_{G'}(S')| \geq \binom{k'}{2} - d' = \binom{k'}{2} - k$. Since $S$ is a clique of size $k$ and $G$ is $k$-partite, $S$ contains exactly one vertex $v_i$ from $V_i$ for each $i \in [1,k]$. Moreover, since each vertex of $\widetilde{K}$ is adjacent to each other vertex in $S'$ and since for each $i \in [1,k]$, $v_i^*$ is adjacent to each other vertex of $S'$ besides $v_i$, $|E_{G'}(S')| = |E_{G'}(K^* \cup \widetilde{K})| + |E_{G'}(K^* \cup \widetilde{K}, S)| + |E_G(S)| = \binom{3k+1}{2} + k \cdot (3k+1) - k + |E_G(S)|$. Since $S$ is a clique of size $k$ in $G$, $|E_{G'}(S')| = \binom{3k+1}{2} + k \cdot (3k+1) - k + \binom{k}{2} = \binom{(3k+1)+k}{2} - k = \binom{k'}{2} - d'$. Hence, $I'$ is a yes-instance of Densest-$k$-Subgraph.

($\Leftarrow$) Let $S'$ be a vertex set of size $k'$ in $G'$ such that $|E_{G'}(S')| \geq \binom{k'}{2} - d' = \binom{4k+1}{2} - k$. Since $G'$ is $(3k+1)$-partite and $S'$ has size $4k+1$, $|E_{G'}(S')| \leq \binom{4k+1}{2} - k$ due to Observation 5.12. Hence, $|E_{G'}(S')| = \binom{4k+1}{2} - k$ and thus, due to Observation 5.12, for each $i \in [1, 3k+1]$, $|S \cap V_i'| \in \{1, 2\}$ and each vertex of $V_i' \cap S'$ is adjacent to each vertex of $S' \setminus V_i'$. Since $S'$ has size $4k+1$ and $V_i'$ has size one for each $i \in [k+1, 3k+1]$, $S'$ contains at least one vertex $v_i$ of $V_i$. Let $S := \{v_i \mid 1 \leq i \leq k\}$. Since $S$ is a subset of $S'$, by the above, $S$ is a clique in $G'$ and thus also a clique in $G$. Hence, $I$ is a yes-instance of Multicolored Clique.

To obtain the further restriction that each vertex $v \in V$ has degree either 1 or $r$, we construct an equivalent instance $I'' := (G'' = (V'', E''), k', d')$ of Densest-$k$-Subgraph as follows. Let $n' := |V'|$. Note that each vertex in $G'$ has degree at most $n' - 1$. We obtain $G''$ by adding for each vertex $v \in V'$, $n' - 1 - |N_{G'}(v)|$ new vertices $X_v$ to $G'$. Additionally, we add edges such that $v$ is the unique neighbor of each vertex of $X_v$ in $G''$ for each vertex $v \in V'$. Hence, each vertex of $V'$ has degree $n' - 1$ in $G''$ and each vertex of $V'' \setminus V'$ has degree one in $G$. It remains to show that $I'$ and $I''$ are equivalent. To this end, note that for each set $S' \subseteq V'$, $|E_{G'}(S')| = |E_{G''}(S')|$. Hence, if $I'$ is a yes-instance of Densest-$k$-Subgraph, then $I''$ is a yes-instance of Densest-$k$-Subgraph. Let $S'' \subseteq V''$ of size $k$ such that $|E_{G''}(S'')| \geq \binom{k'}{2} - d' = \binom{k'}{2} - \frac{k'-1}{4}$. It suffices to show that $S'' \subseteq V'$. Since each vertex of $V'' \setminus V'$ has degree one in $G''$, $|E_{G''}(S'')| \leq \binom{k'}{2} - (k'-2)$ if $S''$ contains at least one vertex of $V'' \setminus V'$. By the fact that $k' = 4k+1 \geq 5$, $k' - 2 > \frac{k'-1}{4}$. Hence, $S'' \subseteq V'$ and thus, $I'$ is a yes-instance of Densest-$k$-Subgraph if $I''$ is a yes-instance of Densest-$k$-Subgraph. □

We are now able to prove Theorem 5.15.

*Proof of Theorem 5.15.* We present a parameter-preserving polynomial-time reduction from Densest-$k$-Subgraph. Hence, due to Lemma 5.16, the W[1]-hardness

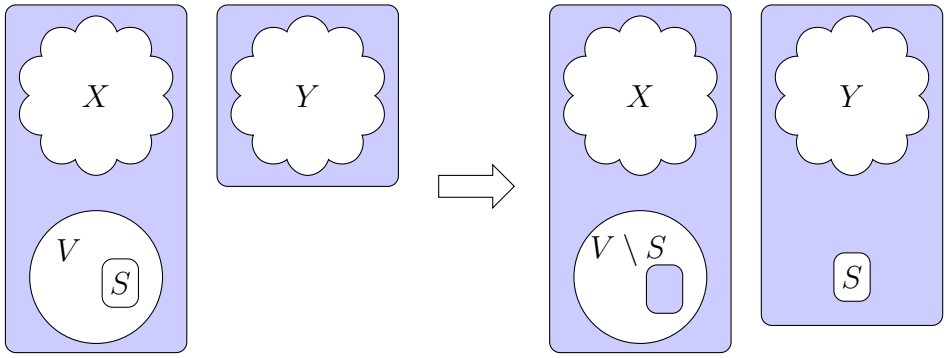



**Figure 5.3:** Two solutions for the instance of LS Cluster Editing constructed in the proof of Theorem 5.15. Left the initial solution. Right an improving solution in the $k$-move neighborhood, if one exists. Such an improving solution only exists if there is a vertex set $S$ of size exactly $k$ in $G$ for which $E(S)$ contains sufficiently many edges.

when parameterized by $k$ and the ETH-based lower bound then directly translate to LS Cluster Editing.

Let $I := (G := (V, E), k, d)$ be an instance of Densest-$k$-Subgraph with $d = \frac{k-1}{4}$ such that $V = V_1 \cup V_r$ such that each vertex of $V_1$ has degree exactly 1 and each vertex of $V_r$ has degree exactly $r$ for some integer $r$. Due to Lemma 5.16, Densest-$k$-Subgraph has the required lower bounds even on such restricted instances. Moreover, let $k \geq 3$. Next we define an instance $I' := (G' := (V', E'), \mathcal{C}, k)$ of LS Cluster Editing with $|\mathcal{C}| = 2$ and show that $I$ is a yes-instance of Densest-$k$-Subgraph if and only if $I'$ is a yes-instance of LS Cluster Editing.

Let $n := |V|$. We obtain $G'$ by adding two cliques $X$ and $Y$ to $G$. The clique $X$ has size $x := n^5 + n - 2r + k - 1$ and we make each vertex of $X$ adjacent to each vertex of $V$. The clique $Y$ has size $y := n^5 = x - (n - 2r + k - 1)$ and we make each vertex of $Y$ adjacent to each vertex of $V_r$. Finally, we define the clustering $\mathcal{C}$ as $\mathcal{C} := \{X \cup V, Y\}$. This completes the construction of $I'$. An example of the initial clustering and an improving clustering (if one exists) is depicted in Figure 5.3. Intuitively, one can only improve over $\mathcal{C}$ by moving a subset $S \subseteq V$ of size $k$ into the cluster containing all vertices of $Y$. Further, this is only improving if $E(S)$ contains at least $\binom{k}{2} - d$ edges.

Before we show that $I$ is a yes-instance of Densest-$k$-Subgraph if and only if $I'$ is a yes-instance of LS Cluster Editing, we first discuss the improvement on the cost of an above describes clustering over the initial clustering.

**Claim 8.** Let $S \subseteq V_r$ and let $\mathcal{C}' := \{X \cup V \setminus S, Y \cup S\}$ be a clustering of $G'$. Then, $\text{cost}(\mathcal{C}) - \text{cost}(\mathcal{C}') = 4 \cdot |E(S)| - |S| \cdot (k-2) - 2\binom{|S|}{2}$.

*Proof of Claim.* Recall that each vertex of $V_r$ (i) has exactly $r$ neighbors in $G$, (ii) exactly $n-r-1$ non-neighbors in $G$, and (iii) is adjacent to each vertex of $X \cup Y$. Hence, by moving a single vertex of $V_r$ from cluster $X \cup V$ to cluster $Y$, the improvement over $\mathcal{C}$ is $-x+y-r+(n-r-1) = y-x+n-2r-1 = -(k-2)$ by definition of $x$ and $y$. Hence, by moving a set $S$ of vertices of $V_r$ from cluster $X \cup V$ to $Y$ gives an improvement over $\mathcal{C}$ of $-|S| \cdot (k-2)$. In this way, we incorrectly charged edge deletions costs for each edge $\{u,v\}$ of $E(S)$ for each of the endpoints $u$ and $v$. Similarly, we incorrectly did not charge edge insertion costs for each non-edge $\{u,v\}$ of $\binom{S}{2} \setminus E(S)$ for each of the endpoints $u$ and $v$. Hence, we obtain the correct improvement by $-|S| \cdot (k-2) + 2 \cdot |E(S)| - 2 \cdot (\binom{|S|}{2} - |E(S)|) = 4 \cdot |E(S)| - |S| \cdot (k-2) - 2\binom{|S|}{2}$. ■

Next, we show that $I$ is a yes-instance of Densest-$k$-Subgraph if and only if $I'$ is a yes-instance of LS Cluster Editing.

($\Rightarrow$) Let $S$ be a set of $k$ vertices of $V$ such that $E(S)$ contains at least $\binom{k}{2} - d = \binom{k}{2} - \frac{k-1}{4}$ edges. Recall that $k \geq 3$. Hence, $S$ contains no vertex of $V_1$, since each vertex of $S$ has at least $k - 1 - \frac{k-1}{4} > 1$ neighbors in $S$.

Consider the clustering $\mathcal{C}' := \{X \cup V \setminus S, Y \cup S\}$ of $G'$. Note that $d_{\text{move}}(\mathcal{C}, \mathcal{C}') \leq k$. Since $S \subseteq V_r$, Claim 8 implies that the improvement of $\mathcal{C}'$ over $\mathcal{C}$ is $4 \cdot |E(S)| - |S| \cdot (k-2) - 2\binom{|S|}{2} \geq 4(\binom{k}{2} - \frac{k-1}{4}) - k \cdot (k-2) - 2\binom{k}{2} = 2 \cdot \binom{k}{2} - (k-1) - (k \cdot (k-1) - k) = k \cdot (k-1) - (k-1) - (k \cdot (k-1) - k) = 1$. Hence, $\mathcal{C}'$ is an improving $k$-move neighbor of $\mathcal{C}$ and thus $I'$ is a yes-instance of LS Cluster Editing.

($\Leftarrow$) Let $\mathcal{C}'$ be a $k$-move neighbor of $\mathcal{C}$ of maximum improvement over $\mathcal{C}$. Suppose that $\mathcal{C}'$ improves over $\mathcal{C}$. In the following, we first show that $\mathcal{C}'$ has size 2. Afterwards, we show that $\mathcal{C}' = \{X \cup V \setminus S, Y \cup S\}$ for some vertex set $S \subseteq V$. Finally, we show that $S$ has size $k$ and that $E(S)$ contains at least $\binom{k}{2} - d = \binom{k}{2} - \frac{k-1}{4}$ edges.

To show that $\mathcal{C}'$ contains exactly two clusters, we first show that $\mathcal{C}'$ contains a cluster $Y'$ with $Y \subseteq Y'$. Let $Y'$ be a cluster of $\mathcal{C}'$ containing the most vertices of $Y$. Since $d_{\text{move}}(\mathcal{C}, \mathcal{C}') \leq k$, $Y'$ contains at least $y - k > k^2$ vertices of $Y'$. Suppose that there is a vertex $v$ of $Y$ such that $v$ is not contained in the cluster $Y'$ and let $C_v$ be the cluster of $\mathcal{C}'$ containing $v$. Note that this implies that $v$ is one of the vertices that moves to a different cluster. By construction, $v$ has at most $|V_r|$ neighbors outside of $Y$. Hence, $C_v$ contains at most $|V_r| + k \leq n + k$ neighbors of $v$ and $Y'$ contains at least $y - k$ neighbors of $v$ and at most $k$ non-neighbors of $v$. Since $y - k > n + k$, the clustering $\mathcal{C}''$ obtained from $\mathcal{C}'$ by removing $v$ from $C_v$ and adding this vertex back to cluster $Y'$ is improving over $\mathcal{C}'$ and has even fewer move-distance from $\mathcal{C}$. Moreover, since $\mathcal{C}'$ is a $k$-move neighbor of $\mathcal{C}$ of maximum improvement over $\mathcal{C}$, no such vertex $v$ of $Y$ exists and thus $Y'$ contains all vertices of $Y$. Similarly, there is a cluster $X'$ of $\mathcal{C}'$ with $X \subseteq X'$. Note that these two clusters $X'$ and $Y'$ are distinct

since $\mathcal{C}'$ is a $k$-move neighbor of $\mathcal{C}$.

Next, we show that each vertex $v$ of $V$ is contained in either cluster $X'$ or in cluster $Y'$. Suppose that this is not the case. Hence, there is a vertex $v$ of $V$ which is contained in a cluster $C_v$ of $\mathcal{C}'$ distinct from $X'$ and $Y'$. Since $d_{\text{move}}(\mathcal{C}, \mathcal{C}') \leq k$, $C_v$ has size at most $k$. Note that $v$ is adjacent to at least $|X' \cap X| = x$ vertices of $X'$ and non-adjacent to at most $|X' \setminus X| \leq n+k$ vertices of $X'$. Hence, the clustering $\mathcal{C}''$ obtained from $\mathcal{C}'$ by removing $v$ from $C_v$ and adding this vertex back to cluster $X$ is improving over $\mathcal{C}'$ and has even fewer move-distance from $\mathcal{C}$. Consequently, since $\mathcal{C}'$ is a $k$-move neighbor of $\mathcal{C}$ of maximum improvement over $\mathcal{C}$, no such vertex $v$ exists. This then implies, that each vertex of $V$ is either contained in cluster $X'$ or contained in cluster $Y'$. Hence, $\mathcal{C}' = \{X', Y'\}$. Moreover, since $X \subseteq X'$ and $Y \subseteq Y'$, there is a vertex set $S \subseteq V$ such that $X' = X \setminus S$ and $Y' = Y \cup S$.

It remains to show that $S$ has size exactly $k$ and that $E(S)$ contains at least $\binom{k}{2} - d$ edges. Note that $S$ has size at most $k$ since $\mathcal{C}'$ is a $k$-move neighbor of $\mathcal{C}$. Ideally, we want to apply Claim 8. Since Claim 8 requires that $S$ contains only vertices of $V_r$, we first have to show that each vertex of $V_1$ is contained in the cluster $X'$. Since each vertex of $V_1$ is adjacent to each vertex of $X$ and non-adjacent to each vertex of $Y$, it is always better not to move a vertex of $V_1$. Hence, $S$ contains only vertices of $V_r$.

Due to Claim 8, the improvement of $\mathcal{C}'$ over $\mathcal{C}$ is $4 \cdot |E(S)| - |S| \cdot (k-2) - 2\binom{|S|}{2}$. Since $\mathcal{C}'$ is improving over $\mathcal{C}$, we get $4 \cdot |E(S)| > |S| \cdot (k-2) + 2\binom{|S|}{2}$ which is equivalent to $|E(S)| > \frac{1}{2} \cdot |S| \cdot \frac{k-2}{2} + \frac{1}{2} \cdot \binom{|S|}{2}$. We consider two cases.

**Case 1:** $|S| < k$**.** If $|S| = k-1$, then the inequality reads as $|E(S)| > \frac{1}{2} \cdot (k-1) \cdot \frac{k-2}{2} + \frac{1}{2} \cdot \binom{k-1}{2} = \binom{k-1}{2} = \binom{|S|}{2}$ which is impossible. For smaller vertex sets $S$, the right side of the inequality even exceeds $\binom{|S|}{2}$.

**Case 2:** $|S| = k$**.** Hence, the inequality reads as $|E(S)| > \frac{1}{2} \cdot k \cdot \frac{k-2}{2} + \frac{1}{2} \cdot \binom{k}{2} = \frac{1}{2} \cdot k \cdot (\frac{k-1}{2} - \frac{1}{2}) + \frac{1}{2} \cdot \binom{k}{2} = \binom{k}{2} - \frac{k}{4}$. This then implies $|E(S)| \geq \binom{k}{2} - \frac{k-1}{4}$.

Hence $S$ has size $k$ and $E(S)$ contains at least $\binom{k}{2} - \frac{k-1}{4}$ edges. Consequently, $I$ is a yes-instance of Densest-$k$-Subgraph.

**Parameter bounds.** Recall that due to Lemma 5.16, Densest-$k$-Subgraph is W[1]-hard when parameterized by $k$ and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. Hence, the above reduction shows that, even if the initial clustering has size two, LS Cluster Deletion is W[1]-hard when parameterized by $k$ and cannot be solved in $f(k) \cdot |V'|^{o(k)}$ time for any computable function $f$, unless the ETH fails, since $|V'| \in n^{\mathcal{O}(k)}$. □

Consequently, due to Theorems 5.14 and 5.15, LS Cluster Editing is W[1]-hard when parameterized by the two arguably most natural parameter combinations $k + \max_{C \in \mathcal{C}} |C|$ and $k + |\mathcal{C}|$.

## 5.4 Algorithms for Permissive Problem variants

In this section, we present our algorithms for the permissive versions of the considered local search problems.

**Parameterization by cluster vertex deletion number and $k$.** In the remainder of the section we provide two algorithms exploiting small modulators to cluster graphs—one for LS Cluster Deletion in Section 5.4.1 and one for LS Cluster Editing in Section 5.4.2. For both algorithms, we use the following notation. We say that a vertex set $M \subseteq V$ is a *(cluster) modulator* of $G$ if $G[V \setminus M]$ is a cluster graph. Let $\mathcal{B}$ be the collection of connected components of $G[V \setminus M]$. We call each clique $B \in \mathcal{B}$ a *bag*. We denote by $\mathrm{cvd}(G)$ the cluster vertex deletion number, that is, the size of a smallest cluster modulator of $G$. If the graph $G$ is clear from the context, we may simply write cvd. An example of a graph $G$ with a cluster modulator $M$ is depicted in Figure 5.4. We may say that a vertex set $M \subseteq V$ is a *2-approximate cluster modulator of* $G$ if $M$ is a cluster modulator of $G$ with $|M| \leq 2 \cdot \mathrm{cvd}(G)$. While it is NP-hard to find a cluster modulator of minimum size [115], a 2-approximate cluster modulator can be found in polynomial time [8].

### 5.4.1 An Algorithm for LS Cluster Deletion

**Lemma 5.17.** *Let $G = (V, E)$ be a graph and let $M$ be a cluster modulator of $G$. Moreover, let $\mathcal{C}_M$ be a given proper clustering of $G[M]$. In $3^{|\mathcal{C}_M|} \cdot n^{\mathcal{O}(1)}$ time, one can find a best clustering $\mathcal{C}$ of $G$ among all proper clusterings $\mathcal{C}'$ of $G$ that extend $\mathcal{C}_M$.*

*Proof.* For each clustering $\mathcal{C}$ of $G$ that extends $\mathcal{C}_M$, the edges between vertices of $M$ in $\mathrm{Cl}(\mathcal{C})$ and $\mathrm{Cl}(\mathcal{C}_M)$ are equal. Consequently, the task can be reformulated as follows: find a clustering of $G$ that maximizes the number of edges having at least one endpoint in $V \setminus M$ among all proper clusterings of $G$ that extend $\mathcal{C}_M$. In the following, we describe a dynamic program solving this reformulated task.

Fix an arbitrary ordering of the bags and let $B_i$ denote the $i$th bag of $\mathcal{B}$ according to this ordering. The dynamic programming table $T$ has entries of type $T[X, i]$ with $X \subseteq \mathcal{C}_M$ and $i \in [0, |\mathcal{B}|]$. For $X \subseteq \mathcal{C}_M$ and $i \in [0, |\mathcal{B}|]$, let $V_X$ denote the union of all clusters of $X$ and let $V_X^i$ denote the union of $V_X$ and the first $i$ bags. The entry $T[X, i]$ stores the maximal number of edges having at least one endpoint in $V_X^i \setminus M$ of any proper clustering of $G[V_X^i]$ that extends $X$. Intuitively, this entry

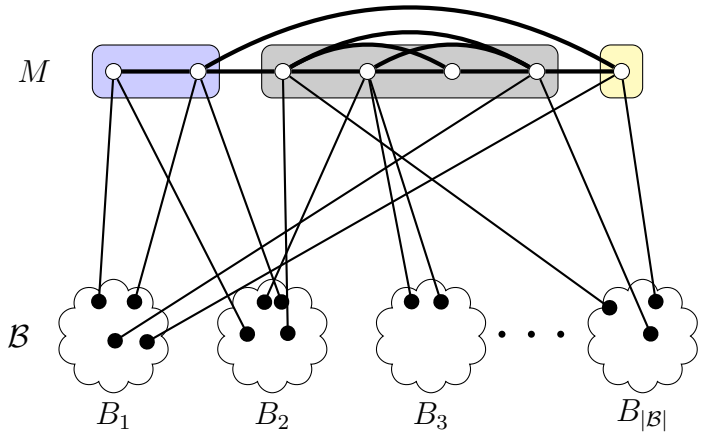



**Figure 5.4:** An example of a graph $G$ with cluster modulator $M$ and bags $\mathcal{B}$ and a proper clustering $\mathcal{C}_M$ of $G[M]$ used as input for the algorithm behind Lemma 5.17. The clouds indicate the bags of $\mathcal{B}$, that is, the maximal cliques of $G - M$, and the colored rectangles indicate the clusters of $\mathcal{C}_M$. Intuitively, the algorithm behind Lemma 5.17 finds an optimal way to distribute the vertices of all bags among the clusters of $\mathcal{C}_M$.

stores the best way to distribute the vertices of the first $i$ bags among the clusters of $X$.

To compute an entry $T[X, i]$, we iterate over all subsets $X'$ of $X$ and check for the best way to distribute the vertices of the first $i-1$ bags among the clusters of $X'$ and the best way to distribute the vertices of the $i$th bag among the clusters of $X \setminus X'$.

Formally, for each $X \subseteq \mathcal{C}_M$, we set $T[X, 0] := 0$ and for each $i \in [1, |\mathcal{B}|]$, we set

$$T[X, i] := \max_{X' \subseteq X} T[X', i-1] + \text{gain}_i^{X \setminus X'}.$$

Here, $\text{gain}_i^{X \setminus X'}$ is the number of edges having at least one endpoint in bag $B_i$ of the best way to distribute the vertices of $B_i$ among the clusters of $X \setminus X'$. This recurrence is correct because no cluster $C$ in any proper clustering $\mathcal{C}$ of $G$ can contain vertices of two distinct bags.

In the following, we describe how to compute the value $\text{gain}_i^Y$ for each $i \in [1, |\mathcal{B}|]$ and each $Y \subseteq \mathcal{C}_M$. Note that computing these values is NP-hard, since Cluster Deletion is NP-hard even on unipolar graphs [24, 110], that is, graphs $G$ where for some vertex set $S$, $G[S]$ is a clique and $G - S$ is a cluster graph. Hence, the algorithm we present take exponential time.

**Claim 9.** In $3^{|\mathcal{C}_M|} \cdot n^{\mathcal{O}(1)}$ time, the values $\text{gain}_i^Y$ can be computed for all $i \in [1, |\mathcal{B}|]$ and all $Y \subseteq \mathcal{C}_M$.

*Proof of Claim.* The computation of this value relies on the following observation: Let $\mathcal{C}$ be a best proper clustering of $G[V_Y \cup B_i]$ that extends $Y$. Moreover, let $C$ be a largest cluster of $\mathcal{C}$. Then $C$ contains all vertices of $B_i$ that are adjacent to all vertices of $C \cap M$. This is true, since if there would be a cluster $C'$ in $\mathcal{C}$ containing a vertex $v$ of $\{v \in B_i \mid C \subseteq N(v)\}$, then $\mathcal{C}' := (\mathcal{C} \setminus \{C, C'\}) \cup \{C \cup \{v\}, C' \setminus \{v\}\}$ is a proper clustering of $G$ that improves over $\mathcal{C}$. The properties of $\mathcal{C}'$ hold since $C$ is a largest cluster of $\mathcal{C}$ and $B_i$ is a clique in $G$.

Hence, to solve this intermediate task, we can branch which cluster $C$ of $Y \cup \{\emptyset\}$ will be extended to be the largest cluster in $\mathcal{C}$, add all vertices of $B_i$ to $C$ that may fit into this cluster and solve the task recursively.

This can also be done by a dynamic program. We introduce the dynamic programming table $D_i$ with entries of type $D_i[Y, Z]$ with $Y \subseteq \mathcal{C}_M$ and $Z \subseteq Y$.

For each set $Z \subseteq Y$, we let $\text{Remain}_Y^Z := B_i \setminus \bigcup_{C \in Y \setminus Z} \{v \in B_i \mid C \subseteq N(v)\}$ denote the set of vertices of $B_i$ that do not fit in any cluster of $Y \setminus Z$. The entry $D_i[Y, Z]$ stores the maximal number of edges having at least one endpoint in $\text{Remain}_Y^Z$ of any proper clustering of $G[\text{Remain}_Y^Z \cup \bigcup_{C \in Y} C]$ that extends $Z$.

For each $Y \subseteq \mathcal{C}_M$, we set $D_i[Y, \emptyset] := \binom{|\text{Remain}_Y^\emptyset|}{2}$ and for each non-empty $Z \subseteq Y$, we set

$$D_i[Y, Z] := \max\left(\binom{|\text{Remain}_Y^Z|}{2}, \max_{C \in Z}\left(\binom{|\text{Fit}_C|}{2} + |\text{Fit}_C| \cdot |C| + D_i[Y, Z \setminus \{C\}]\right)\right),$$

where $\text{Fit}_C := \{v \in \text{Remain}_Y^Z \mid C \subseteq N(v)\}$ denotes the set of vertices of $\text{Remain}_Y^Z$ that fit into the cluster $C$.

This recurrence is correct by the above observation: In each optimal proper clustering $\mathcal{C}_Z^i$ of $G[\text{Remain}_Y^Z \cup \bigcup_{C \in Y} C]$ that extends $Z$, there is a largest cluster $C' \in \mathcal{C}_Z^i$ that contains all vertices of $\text{Fit}_{C' \cap M}$.

Finally, for each $Y \subseteq \mathcal{C}_M$, we set $\text{gain}_i^Y := D_i[Y, Y]$. Since for each $i \in [1, |\mathcal{B}|]$, the table $D_i$ contains $3^{|\mathcal{C}_M|}$ entries and each such entry can be computed in $n^{\mathcal{O}(1)}$ time, the values $\text{gain}_i^Y$ can be computed for all $i \in [1, |\mathcal{B}|]$ and all $Y \subseteq \mathcal{C}_M$ in the stated running time.

Moreover, note that a corresponding clustering can be computed via traceback in the same asymptotic running time. ■

Let $\mathcal{C}^*$ be any best clustering of $G$ among all proper clusterings of $G$ that extend $\mathcal{C}_M$. Then, the number of edges of $\text{Cl}(\mathcal{C}^*)$ having at least one endpoint in any bag is stored in $T[\mathcal{C}_M, |\mathcal{B}|]$. Moreover, a corresponding clustering can be found via traceback.

Since for each $i \in [0, |\mathcal{B}|]$ and each $X \subseteq \mathcal{C}_M$, the table entry $T[X, i]$ can be computed in $2^{|X|} \cdot n^{\mathcal{O}(1)}$ time and there are $2^{|\mathcal{C}_M|}$ choices for $X$, each table entry of $T$ can be computed in time $\sum_{i=1}^{|\mathcal{C}_M|} \binom{|\mathcal{C}_M|}{i} \cdot 2^i \cdot n^{\mathcal{O}(1)} \subseteq 3^{|\mathcal{C}_M|} \cdot n^{\mathcal{O}(1)}$. □

Based on Lemma 5.17 we can obtain the following FPT-algorithm for Cluster Deletion when parameterized by cvd: First, compute a minimum cluster modulator $M$ of $G$ in $1.811^{\text{cvd}} \cdot n^{\mathcal{O}(1)}$ time [162]. Second, iterate over all possible clusterings of $G[M]$ and apply the algorithm behind Lemma 5.17. This implies a running time of $3^{\text{cvd}} \cdot \mathbf{B}_{\text{cvd}} \cdot n^{\mathcal{O}(1)}$, where $\mathbf{B}_{\text{cvd}}$ is the cvd-th Bell number which denotes the number of partitions of a set of size cvd. Since for each $n \in \mathbb{N}$, $\mathbf{B}_n < (\frac{n}{\ln(n+1)})^n$ [16], this implies the following.

**Theorem 5.18.** Cluster Deletion *can be solved in* $\text{cvd}^{\text{cvd}} \cdot n^{\mathcal{O}(1)}$ *time.*

Next, we use Lemma 5.17 to obtain a permissive algorithm for LS Cluster Deletion when parameterized by cvd and $k$.

**Theorem 5.19.** *Let $G$ be a graph and let $M$ be a given cluster modulator of $G$. Moreover, let $\mathcal{C}$ be a proper clustering of $G$ and let $k \in \mathbb{N}$. In $|M|^{2k} \cdot \binom{|M|}{k} \cdot k^k \cdot 3^{4k} \cdot n^{\mathcal{O}(1)}$ time, one can find a proper clustering $\mathcal{C}'$ of $G$ which is at least as good as a best proper clustering $\mathcal{C}^*$ of $G$ with $d_{\text{move}}(\mathcal{C}, \mathcal{C}^*) \leq k$.*

*Proof.* Let $\mathcal{C}^*$ be a clustering of $G$ that maximizes $|\text{Cl}(\mathcal{C}^*)|$ among all proper clusterings $\mathcal{C}''$ of $G$ with $d_{\text{move}}(\mathcal{C}, \mathcal{C}'') \leq k$. Moreover, let $\mathcal{M} := \{C \in \mathcal{C} \mid C \cap M \neq \emptyset\}$ and $\mathcal{M}^* := \{C \in \mathcal{C}^* \mid C \cap M \neq \emptyset\}$ denote the sets of clusters intersecting $M$, of $\mathcal{C}$ and $\mathcal{C}^*$, respectively.

Let $\mathcal{C}_M := \{C \cap M \mid C \in \mathcal{M}\}$ denote the clusters of $\mathcal{M}$ restricted to the vertices of $M$. Similarly, let $\mathcal{C}^*_M := \{C \cap M \mid C \in \mathcal{M}^*\}$. Note that $d_{\text{move}}(\mathcal{C}, \mathcal{C}^*) \leq k$ implies $d_{\text{move}}(\mathcal{C}_M, \mathcal{C}^*_M) \leq k$. Hence, to find a proper clustering $\mathcal{C}'$ of $G$ that is at least as good as $\mathcal{C}^*$, it suffices to enumerate all proper clusterings $\mathcal{C}'_M$ of $G[M]$ with $d_{\text{move}}(\mathcal{C}_M, \mathcal{C}'_M) \leq k$ (which includes $\mathcal{C}^*_M$) and to compute, for each such clustering $\mathcal{C}'_M$, any best proper clustering of $G$ that extends $\mathcal{C}'_M$. As the latter task can be done in $3^{|\mathcal{C}'_M|} \cdot n^{\mathcal{O}(1)}$ time by Lemma 5.17, it remains to describe how one can enumerate all such proper clusterings $\mathcal{C}'_M$ of $G[M]$.

This can be done in $\binom{|M|}{k} \cdot (|\mathcal{M}| + k)^k \cdot n^{\mathcal{O}(1)}$ time by iterating over all possible subsets $M' \subseteq M$ of size $k$ and iterating over all possible ways to move these $k$ vertices into any of the clusters of $\mathcal{M}$ (including the clusters where these vertices came from) or opening a new cluster.

Hence, this algorithm runs in $\binom{|M|}{k} \cdot (|\mathcal{M}| + k)^k \cdot 3^{|\mathcal{M}|+k} \cdot n^{\mathcal{O}(1)}$ time. Note that this is not the desired running time since $|\mathcal{M}|$ occurs in the exponent of the running time and might be much larger than $k$.

To obtain the desired running time, we perform some initial branching if $|\mathcal{M}| > 2k$. The idea behind this initial branching relies on Observation 5.2. Intuitively, Observation 5.2 states that at least $|\mathcal{M}| - 2k$ clusters are identical in $\mathcal{M}^*$ and $\mathcal{M}$ since at most $k$ vertices were moved to obtain $\mathcal{M}^*$ from $\mathcal{M}$. That is, $|\mathcal{M} \cap \mathcal{M}^*| \geq |\mathcal{M}| - 2k$. In other words, there is a subset $\mathcal{M}' \subseteq \mathcal{M}$ of size at most $2k$ such that $\mathcal{M} \setminus \mathcal{M}' \subseteq \mathcal{M} \cap \mathcal{M}^*$. This implies that all edge deletions having at least one endpoint in any cluster of $\mathcal{M} \setminus \mathcal{M}'$ are identical in $\mathrm{Cl}(\mathcal{C})$ and $\mathrm{Cl}(\mathcal{C}^*)$. Hence, by applying for each subset $\mathcal{M}' \subseteq \mathcal{M}$ of size at most $2k$ the above described algorithm on the graph $G[V \setminus \cup_{C \in \mathcal{M} \setminus \mathcal{M}'} C]$, we find a proper clustering of $G$ which is at least as good as $\mathcal{C}^*$. This initial branching can be done in $|\mathcal{M}|^{2k} \cdot n^{\mathcal{O}(1)} \subseteq |M|^{2k} \cdot n^{\mathcal{O}(1)}$ time.

Since for each such branching-instance, there are at most $2k$ clusters containing vertices of $M$, the total running time evaluates to $|M|^{2k} \cdot \binom{|M|}{k} \cdot (3k)^k \cdot 3^{3k} \cdot n^{\mathcal{O}(1)} = |M|^{2k} \cdot \binom{|M|}{k} \cdot k^k \cdot 3^{4k} \cdot n^{\mathcal{O}(1)}$ time. □

Since a 2-approximate cluster modulator can be found in polynomial time [8], Theorem 5.19 implies the following:

**Theorem 5.20.** *The permissive version of* LS Cluster Deletion *can be solved in* $(k^k + 2^{\mathcal{O}(k)} \cdot \mathrm{cvd}^{3k}) \cdot n^{\mathcal{O}(1)}$ *time.*

*Proof.* Let $I := (G = (V, E), k, \mathcal{C})$ be an instance of LS Cluster Deletion. First, check in $1.811^k \cdot n^{\mathcal{O}(1)}$ time, whether $G$ has a cluster modulator of size at most $k$ [162]. If this is the case, one can find an optimal proper clustering $\mathcal{C}'$ of $G$ in time $\mathrm{cvd}^{\mathrm{cvd}} \cdot n^{\mathcal{O}(1)} \subseteq k^k \cdot n^{\mathcal{O}(1)}$ due to Theorem 5.18. Otherwise, $k < \mathrm{cvd}$. In this case, we compute a 2-approximate cluster modulator $M$ in polynomial time [8] and find a proper clustering $\mathcal{C}'$ of $G$ which is at least as good as a best proper clustering $\mathcal{C}^*$ of $G$ with $d_{\mathrm{move}}(\mathcal{C}, \mathcal{C}^*) \leq k$ in time $|M|^{2k} \cdot \binom{|M|}{k} \cdot k^k \cdot 3^{4k} \cdot n^{\mathcal{O}(1)}$ due to Theorem 5.19. Since $k < \mathrm{cvd} \leq |M|$, we get that $\binom{|M|}{k} \cdot k^k \leq |M|^k \cdot \frac{k^k}{k!} \leq |M|^k \cdot 2^{\mathcal{O}(k)}$ due to Stirling's approximation. Hence, the running time of the case $k < \mathrm{cvd}$ evaluates to $2^{\mathcal{O}(k)} \cdot |M|^{3k} \cdot n^{\mathcal{O}(1)} \subseteq 2^{\mathcal{O}(k)} \cdot (2 \cdot \mathrm{cvd})^{3k} \cdot n^{\mathcal{O}(1)} = 2^{\mathcal{O}(k)} \cdot \mathrm{cvd}^{3k} \cdot n^{\mathcal{O}(1)}$ time. In both cases, the running time is upper-bounded by $(k^k + 2^{\mathcal{O}(k)} \cdot \mathrm{cvd}^{3k}) \cdot n^{\mathcal{O}(1)}$ time. □

Note that this running time is essentially optimal with respect to the presented ETH based lower bound from Theorem 5.7.

### 5.4.2 An Algorithm for LS Cluster Editing

In this subsection, we present a permissive algorithm for LS Cluster Editing with a running time similar to the one for LS Cluster Deletion presented in Theorem 5.20.

**Theorem 5.21.** *Let $G = (V, E)$ be a graph, let $\mathcal{C}$ be a clustering of $G$, and let $k \in \mathbb{N}$. In $2^{\mathcal{O}(k)} \cdot k^k \cdot \mathrm{cvd}^{3k} \cdot n^{\mathcal{O}(1)}$ time, one can find a clustering that improves over $\mathcal{C}$ or correctly output that there is no clustering $\mathcal{C}'$ of $G$ with $d_{\mathrm{move}}(\mathcal{C}, \mathcal{C}') \leq k$ that improves over $\mathcal{C}$.*

To give a better intuition for the following algorithm, we switch to the interpretation of LS Cluster Editing where the initial solution is a cluster coloring $\chi_{\mathcal{C}}$ of $\mathcal{C}$. That is, if the given coloring $\chi_{\mathcal{C}}$ can be improved within the $k$-move neighborhood, then we need to find *any* coloring $\chi^*$ improving over $\chi_{\mathcal{C}}$. This coloring $\chi^*$ is not required to be in the $k$-move neighborhood of $\chi_{\mathcal{C}}$.

Let $I := (G = (V, E), k, \chi_{\mathcal{C}})$ be an instance of LS Cluster Editing, let $M$ be a cluster modulator of $G$. Let $\alpha \in \mathbb{N}$ be a color used by $\chi_{\mathcal{C}}$. We say that $\alpha$ is a *modulator color* if $\chi_{\mathcal{C}}^{-1}(\alpha) \cap M \neq \emptyset$. Otherwise, we say that $\alpha$ is a *bag color*. In the remainder of this section, we denote by $\mathrm{col}_{\mathrm{Mod}}$ and $\mathrm{col}_{\mathrm{Bag}}$ the set of modulator colors and bag colors of $\chi_{\mathcal{C}}$, respectively. Note that these color sets are defined with respect to the initial coloring $\chi_{\mathcal{C}}$ of the LS Cluster Editing-instance $I$. Recall that each bag $B \in \mathcal{B}$ is a clique in $G$ and a connected component of $G[V \setminus M]$. Fix an arbitrary ordering of the bags and let $B_i$ denote the $i$th bag of $\mathcal{B}$ according to this ordering.

We first observe that we can improve the initial coloring in polynomial time if at least one of the following cases applies.

**Observation 5.22.** *If there is a bag color $\alpha$ such that vertices of two distinct bags receive color $\alpha$ under $\chi_{\mathcal{C}}$, then one can find a coloring $\chi'$ of $V$ in polynomial time that improves over $\chi_{\mathcal{C}}$.*

Recall that Observation 5.3 states that a clustering is not 1-optimal if there is a cluster of diameter larger than 2. Hence, Observation 5.22 is directly implied by Observation 5.3.

**Observation 5.23.** *If there is a bag $B \in \mathcal{B}$ such that two vertices of $B$ receive distinct bag colors under $\chi_{\mathcal{C}}$, then one can find a coloring $\chi'$ of $V$ that improves over $\chi_{\mathcal{C}}$ in polynomial time.*

Intuitively, this observation holds, since one can improve over $\chi_{\mathcal{C}}$ by moving all vertices of $B$ that receive a bag color under $\chi_{\mathcal{C}}$ to a joined color. Since all these vertices of $B$ are pairwise adjacent, this new coloring then uses fewer edge deletions and the same number of edge insertions.

Hence, we assume in the following, that for each bag color $\alpha \in \mathrm{col}_{\mathrm{Bag}}$, $\chi_{\mathcal{C}}^{-1}(\alpha)$ contains only vertices of a single bag and that for each bag $B_i \in \mathcal{B}$, there is at most one bag color $\alpha_i \in \mathrm{col}_{\mathrm{Bag}}$ with $\chi_{\mathcal{C}}^{-1}(\alpha_i) \subseteq B_i$.

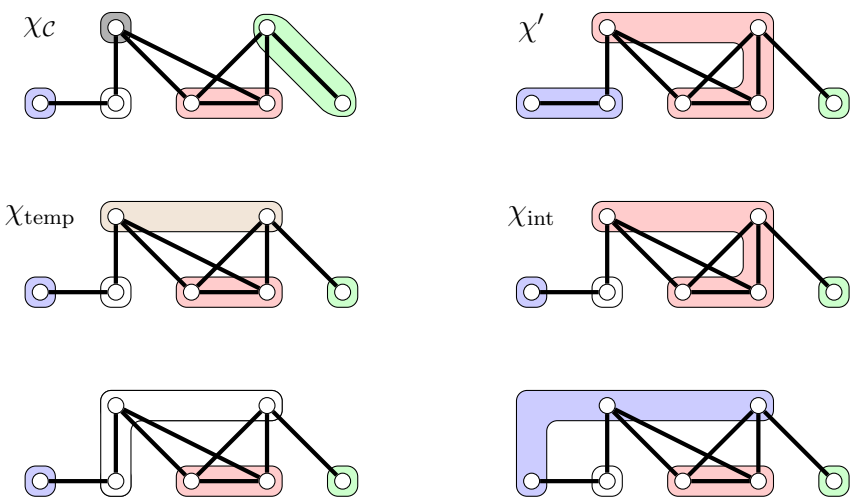



**Figure 5.5:** An example of the different types of considered colorings: The initial clustering $\chi_{\mathcal{C}}$, a 3-move neighbor $\chi'$ of $\chi_{\mathcal{C}}$, the intermediate coloring $\chi_{\text{int}}$ for $\chi'$, and a quasi-intermediate template coloring $\chi_{\text{temp}}$ for $\chi'$. The upper two vertices represent the considered cluster modulator. Gray and green are the modulator colors, blue, white, and red are the bag colors, and brown is the unique moving color of $\chi_{\text{temp}}$. The bottom two colorings are the only pairwise non-isomorphic colorings beside $\chi_{\text{temp}}$ and $\chi_{\text{int}}$ that can be achieved by applying a template recolor-function to $\chi_{\text{temp}}$. Recoloring the moving color brown to green is not possible by a template recolor-function, since green is a modulator color.

Moreover, we assume that $\text{col}_{\text{Mod}}$ has size $\mathcal{O}(k)$. In the final algorithm we use an initial branching—similar to the one used in the algorithm behind Theorem 5.19—to ensure that this assumption is fulfilled.

Let $\chi'$ be a coloring of $V$ and let $\chi_{\text{int}}$ be the coloring that agrees with $\chi_{\mathcal{C}}$ on all vertices of $V \setminus M$ and with $\chi'$ on all vertices of $M$. We call $\chi_{\text{int}}$ the *intermediate coloring* for $\chi'$. The idea behind this definition is the following.

**Observation 5.24.** *Let $\chi'$ be a coloring of $V$ and let $\chi_{\text{int}}$ be the intermediate coloring for $\chi'$. It holds that $d_{\text{move}}(\chi_{\mathcal{C}}, \chi_{\text{int}}) + d_{\text{move}}(\chi_{\text{int}}, \chi') = d_{\text{move}}(\chi_{\mathcal{C}}, \chi')$.*

Suppose that there is an improving $k$-move neighbor $\chi'$ of $\chi_{\mathcal{C}}$. Due to Observation 5.24, to find a coloring of $V$ that improves over $\chi_{\mathcal{C}}$, it is sufficient to do the following: Iterate over all colorings $\chi_{\text{int}}$ with $d_{\text{move}}(\chi_{\mathcal{C}}, \chi_{\text{int}}) \leq k$ that agree with $\chi_{\mathcal{C}}$ on all vertices of $V \setminus M$. For each such coloring $\chi_{\text{int}}$ check whether there is a coloring $\chi'$ that improves over $\chi_{\mathcal{C}}$ with $d_{\text{move}}(\chi_{\text{int}}, \chi') \leq k - d_{\text{move}}(\chi_{\mathcal{C}}, \chi_{\text{int}})$ such that $\chi_{\text{int}}$ and $\chi'$ agree on all vertices of $M$, that is, where $\chi_{\text{int}}$ is the intermediate coloring for $\chi'$. Unfortunately, such an approach exceeds the desired running time for our

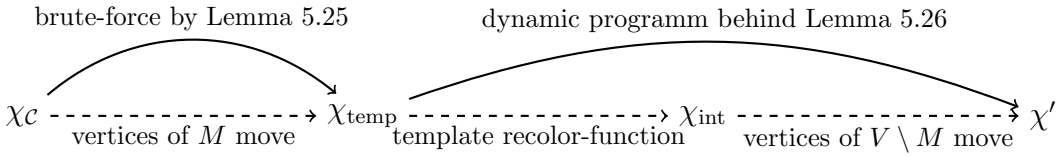



**Figure 5.6:** An overview over the four different kinds of considered colorings. For the initial coloring $\chi_{\mathcal{C}}$ and an improving coloring $\chi'$ with $d_{\text{move}}(\chi_{\mathcal{C}}, \chi') \leq k$, there is a template coloring $\chi_{\text{temp}}$ and an intermediate coloring $\chi_{\text{int}}$ such that there is a template recolor-function between $\chi_{\text{temp}}$ and $\chi_{\text{int}}$. To find a coloring at least as good as $\chi'$, we first iterate over all possible choices of the template coloring $\chi_{\text{temp}}$ and afterwards search for the best coloring for which $\chi_{\text{temp}}$ is quasi-intermediate. This is done by a dynamic program that simultaneously finds the best template recolor-function and the best way to distribute the bag vertices by using at most $k$ moves.

algorithm since there are $n^{\mathcal{O}(k)}$ possibilities for the colorings $\chi_{\text{int}}$ because there may be up to $\Theta(n)$ bag colors and each vertex of $M$ may receive any color. To obtain the desired running time, we instead only iterate over "template colorings". Here, a coloring $\chi_{\text{temp}}$ of $V$ is a *template coloring* if

- $d_{\text{move}}(\chi_{\mathcal{C}}, \chi_{\text{temp}}) \leq k$,
- $\chi_{\text{temp}}$ agrees with $\chi_{\mathcal{C}}$ on all vertices of $V \setminus M$, and
- no vertex of $M$ receives a bag color under $\chi_{\text{temp}}$.

For a template coloring $\chi_{\text{temp}}$, let $\text{col}_{\text{Move}}$ denote the colors of $\mathbb{N} \setminus (\text{col}_{\text{Mod}} \cup \text{col}_{\text{Bag}})$ that are used by $\chi_{\text{temp}}$. We call the colors of $\text{col}_{\text{Move}}$ the *moving colors* of $\chi_{\text{temp}}$. Note that only vertices of $M$ may receive a moving color under $\chi_{\text{temp}}$ and that there are at most $k$ moving colors.

Figure 5.5 depicts an example for the types of colorings used in this algorithm, and Figure 5.6 illustrates how we use these colorings to find a coloring that improves over $\chi_{\mathcal{C}}$, if an improving coloring exists in the $k$-move neighborhood of $\chi_{\mathcal{C}}$.

The idea behind template colorings is that a template coloring $\chi_{\text{temp}}$ may represent intermediate colorings for many colorings $\chi'$ in the following way: For a coloring $\chi'$ of $V$, we say that a template coloring $\chi_{\text{temp}}$ is *quasi-intermediate for* $\chi'$ if there is a "template recolor-function" $f \colon \mathbb{N} \to \mathbb{N}$ such that $f \circ \chi_{\text{temp}}$ is the intermediate coloring for $\chi'$. Herein, a function $f \colon \mathbb{N} \to \mathbb{N}$ is a *template recolor-function* if $f$ preserves identity on all colors of $\mathbb{N} \setminus \text{col}_{\text{Move}}$ and where $f|_{\text{col}_{\text{Move}}}$ maps each color of $\text{col}_{\text{Move}}$ to some color of $\mathbb{N} \setminus \text{col}_{\text{Mod}}$ injectively. Essentially, this means that each of the moving colors of $\chi_{\text{temp}}$ may be identified with any bag color. Informally, this is due to

the fact that each vertex that receives a moving color under $\chi_{\text{temp}}$ already changed its color and we may move all vertices of that moving color together to any bag color while preserving the move-distance to $\chi_{\mathcal{C}}$. Note that, for each coloring $\chi'$ of $V$ with $d_{\text{move}}(\chi_{\mathcal{C}}, \chi') \leq k$, there is a template coloring $\chi_{\text{temp}}$ which is quasi-intermediate for $\chi'$. In contrast to intermediate colorings, we can enumerate a maximal set $\mathcal{X}$ of pairwise non-isomorphic template colorings in the desired running time.

**Lemma 5.25.** *One can compute a maximal set $\mathcal{X}$ of pairwise non-isomorphic template colorings in time $(|\text{col}_{\text{Mod}}| + k)^k \cdot |M|^k \cdot n^{\mathcal{O}(1)}$.*

*Proof.* Recall that for each template coloring $\chi_{\text{temp}}$, $d_{\text{move}}(\chi_{\mathcal{C}}, \chi_{\text{temp}}) \leq k$. Moreover, by the definition of a template coloring, $D_{\text{Move}}(\chi_{\mathcal{C}}, \chi_{\text{temp}}) \subseteq M$. Hence, to obtain a maximal set $\mathcal{X}$ of pairwise non-isomorphic template colorings, consider all possible subsets of $M$ of size at most $k$ and consider all possible ways of assigning colors of $\text{col}_{\text{Mod}} \cup A$ to these at most $k$ vertices, where $A$ is an arbitrary set of $k$ colors from $\mathbb{N} \setminus (\text{col}_{\text{Mod}} \cup \text{col}_{\text{Bag}})$. Note that this can be done in the stated running time. □

In order to find a coloring $\chi^*$ of $V$ that improves over $\chi_{\mathcal{C}}$ (provided that there is such a coloring in the $k$-move neighborhood of $\chi_{\mathcal{C}}$), it suffices to do the following: For each template coloring $\chi_{\text{temp}}$ in a maximal set of pairwise non-isomorphic template colorings, find the best coloring $\chi'$ in the $k$-move neighborhood of $\chi_{\mathcal{C}}$ such that $\chi_{\text{temp}}$ is quasi-intermediate for $\chi'$. In the following, we show that the latter task can be done in $2^{\mathcal{O}(|\text{col}_{\text{Mod}}|+k)} \cdot n^{\mathcal{O}(1)}$ time for each individual template coloring.

**Lemma 5.26.** *Let $\chi_{\text{temp}}$ be a template coloring. In $2^{\mathcal{O}(|\text{col}_{\text{Mod}}|+k)} \cdot n^{\mathcal{O}(1)}$ time, one can find a coloring of $V$ that improves over $\chi_{\mathcal{C}}$, or correctly output that the $k$-move neighborhood of $\chi_{\mathcal{C}}$ does not contain a coloring $\chi'$ that improves over $\chi_{\mathcal{C}}$ and where $\chi_{\text{temp}}$ is quasi-intermediate for $\chi'$.*

Before we provide the proof of Lemma 5.26, we introduce some notation on functions we use to describe our algorithm. For $x \in \mathbb{N}$ and $y \in \mathbb{N}$, we denote by $\text{id}_{[x \mapsto y]}$ the function that preserves identity on $\mathbb{N} \setminus \{x\}$ and where $\text{id}_{[x \mapsto y]}(x) := y$. For functions $\widehat{\omega} \colon A \to \mathbb{Z}$ and $\widetilde{\omega} \colon B \to \mathbb{Z}$ with $A \subseteq B$, we denote by $\widehat{\omega} - \widetilde{\omega}$ the function $\bar{\omega} \colon A \to \mathbb{Z}$, where for each $a \in A$, $\bar{\omega}(a) := \widehat{\omega}(a) - \widetilde{\omega}(a)$. Similarly, we denote by $\widehat{\omega} + \widetilde{\omega}$ the function $\bar{\omega} \colon A \to \mathbb{Z}$, where for each $a \in A$, $\bar{\omega}(a) := \widehat{\omega}(a) + \widetilde{\omega}(a)$.

*Proof.* Let $\text{col}_{\text{Move}}$ be the set of moving colors of $\chi_{\text{temp}}$. Suppose that $\chi_{\text{temp}}$ is quasi-intermediate for some coloring $\chi'$ of $V$ with $d_{\text{move}}(\chi_{\mathcal{C}}, \chi') \leq k$ that improves over $\chi_{\mathcal{C}}$. Hence, there is a template recolor-function $f_{\text{int}}$ such that $\chi_{\text{int}} := f_{\text{int}} \circ \chi_{\text{temp}}$ is the intermediate coloring for $\chi'$.

In the following, we describe an algorithm that finds a coloring $\chi^*$ which is at least as good as $\chi'$ such that $\chi_{\text{temp}}$ is quasi-intermediate for $\chi^*$. The main idea of this algorithm relies on the following fact: Since $d_{\text{move}}(\chi_{\mathcal{C}}, \chi') \leq k$, at most $k$ bag vertices may change their color. Hence, for each vertex set $A \subseteq V \setminus M$:

$$\sum_{i \in \mathbb{N}} \big|\, |A \cap {\chi'}^{-1}(i)| - |A \cap {\chi_{\mathcal{C}}}^{-1}(i)| \,\big| \leq 2k, \tag{5.1}$$

$$\sum_{i \in \mathbb{N}} \big(|A \cap {\chi'}^{-1}(i)| - |A \cap {\chi_{\mathcal{C}}}^{-1}(i)|\big) = 0. \tag{5.2}$$

In the following, we define functions that describe these properties. Let $V' \subseteq V \setminus M$ be a set of vertices and let $\psi$ be a coloring of $V$ with $\psi = f \circ \chi_{\text{temp}}$ for some template recolor-function $f$. Moreover, let $X \subseteq \mathbb{N}$ be a set of colors with $V' \subseteq \psi^{-1}(X)$, that is, a set of colors that contains at least those colors that are assigned to the vertices of $V'$ by $\psi$. A *resize function* for $V'$ and $\psi$ is a function $\omega \colon X \to \mathbb{Z}$ satisfying

$$\sum_{i \in X} |\omega(i)| \leq 2k, \quad \sum_{i \in X} \omega(i) = 0, \quad \text{and} \quad |V' \cap \psi^{-1}(i)| + \omega(i) \geq 0 \text{ for each } i \in X.$$

A coloring $\widehat{\psi}$ of $V$ *fits* $\omega$ and $\psi$ if (a) for each color $i \in X$, $|V' \cap \psi^{-1}(i)| + \omega(i) = |V' \cap \widehat{\psi}^{-1}(i)|$ and (b) $\psi$ is the intermediate coloring for $\widehat{\psi}$. Further, $\omega$ is called a $(\psi, \widehat{\psi})$-*fitting* resize function in this case. Intuitively, $\omega(i)$ denotes the change between $\psi$ and $\widehat{\psi}$ in the number of vertices of $V'$ that receive color $i$. Hence, the last condition of a resize function describes that one cannot decrease the number of vertices of $V'$ in a cluster by more than the number of vertices of $V'$ currently in that cluster. Note that for any coloring $\chi^*$ for which $\chi_{\text{temp}}$ is quasi-intermediate, there is a unique template recolor-function $f^*$ and that for each vertex set $V' \subseteq V \setminus M$, all $(f^* \circ \chi_{\text{temp}}, \chi^*)$-fitting resize functions for $V'$ and $f^* \circ \chi_{\text{temp}}$ agree on their non-zero entries.

Recall that $d_{\text{move}}(\chi_{\mathcal{C}}, \chi') \leq k$. Thus, there is a $(\chi_{\text{int}}, \chi')$-fitting resize function $\omega' \colon \mathbb{N} \to \mathbb{Z}$ for $V \setminus M$ and $\chi_{\text{int}}$. Moreover, for each $i \in [1, |\mathcal{B}|]$ there is a $(\chi_{\text{int}}, \chi')$-fitting resize function $\omega'_i \colon \mathbb{N} \to \mathbb{Z}$ for $B_i$ and $\chi_{\text{int}}$ such that $\omega'(\alpha) = \sum_{i \in [1, |\mathcal{B}|]} \omega'_i(\alpha)$ for each color $\alpha \in \mathbb{N}$. The motivation behind fitting resize functions is the following: For a given template recolor-function $f$ and a given resize function $\omega_i$ for $B_i$ and $f \circ \chi_{\text{temp}}$, one can find in polynomial time a coloring $\psi$ of $V$ that minimizes the required number of edge-modifications with one endpoint in $B_i$ and the other endpoint in $B_i \cup M$ among all colorings $\psi$ of $V$ that fit $\omega_i$ and $f \circ \chi_{\text{temp}}$. In the following, we show how this can be done.

**Claim 10.** Let $\psi$ be a coloring of $V$ with $\psi = f \circ \chi_{\text{temp}}$ for some template recolor-function $f$, let $B_i \in \mathcal{B}$ be a bag, and let $X \subseteq \mathbb{N}$ be a set of $\mathcal{O}(n)$ colors such

that $B_i \subseteq \psi^{-1}(X)$. Moreover, let $\widetilde{\omega}_i \colon X \to \mathbb{Z}$ be a resize function for $B_i$ and $\psi$. In $n^{\mathcal{O}(1)}$ time, one can compute the minimum number of required edge-modifications with one endpoint in $B_i$ and the other in $B_i \cup M$ among all colorings $\widetilde{\psi}$ of $V$ that fit $\widetilde{\omega}_i$ and $\psi$.

*Proof of Claim.* For each color $\alpha \in X$, let $n_\alpha := |B_i \cap \psi^{-1}(\alpha)| + \widetilde{\omega}_i(\alpha)$ denote the number of vertices in $B_i$ receiving color $\alpha$ in any coloring $\widetilde{\psi}$ fulfilling the restrictions of the claim. Recall that each vertex of $B_i$ receives a color of $X$ under $\psi$ and that $\widetilde{\omega}_i$ is a resize function for $B_i$ and $\psi$. Hence, $\sum_{\alpha \in X} n_\alpha = |B_i|$.

The task is to find a coloring $\widetilde{\psi}$ of $V$ that has a minimum number of edge-modifications with one endpoint in $B_i$ and the other endpoint in $B_i \cup M$, such that $\widetilde{\psi}$ and $\psi$ agree on all vertices of $M$ and for each color $\alpha \in X$,

$$|B_i \cap \widetilde{\psi}^{-1}(\alpha)| = |B_i \cap \psi^{-1}(\alpha)| + \widetilde{\omega}_i(\alpha) = n_\alpha.$$

Note that, based on the last condition, the number of edge-modifications having both endpoints in $B_i$ is equal for all colorings $\widetilde{\psi}$ fulfilling all these conditions. The number of edge-modifications having both endpoints in $B_i$ is $\frac{1}{2} \cdot \sum_{\alpha \in X} n_\alpha \cdot (|B_i| - n_\alpha)$.

Hence, in the following, we only have to determine the minimum number of edge-modifications having one endpoint in $B_i$ and the other endpoint in $M$ of any sought coloring $\widetilde{\psi}$. We show that this can be done by finding a minimum-weight perfect matching in an auxiliary bipartite graph, which can be done in polynomial time [112].

We define an auxiliary weighted complete bipartite graph $G' = (V', E')$ with bipartition $(B_i, V_X)$, where $V_X := \cup_{\alpha \in X} V_\alpha$ and, for each color $\alpha \in X$, $V_\alpha$ is an arbitrary vertex set of size $n_\alpha$. Note that $|V_X| = \sum_{\alpha \in X} n_\alpha = |B_i|$ since $\widetilde{\omega}_i$ is a resize function for $B_i$ and $\psi$. It remains to define the edge-weights of $G'$. For each color $\alpha \in X$, each vertex $u_\alpha \in V_\alpha$, and each vertex $v \in B_i$, we set the weight of the edge $\{v, u_\alpha\}$ to $|(\psi^{-1}(\alpha) \cap M) \setminus N_G(v)| + |(M \setminus \psi^{-1}(\alpha)) \cap N_G(v)|$, that is, the number of non-neighbors of $v$ in $M$ that receive color $\alpha$ under $\psi$ plus the number of neighbors of $v$ in $M$ that do not receive color $\alpha$ under $\psi$.

Note that the edge weights are set up such that a minimum-weight perfect matching in $G'$ has weight equal to the minimum number of edge-modifications having one endpoint in $B_i$ and one endpoint in $M$ of any sought coloring $\widetilde{\psi}$ of $V$. ■

In the following, let $\mathrm{col}_{\mathrm{M}} := \mathrm{col}_{\mathrm{Mod}} \cup \mathrm{col}_{\mathrm{Move}}$. Note that for a template recolor-function $f^*$, an $(f^* \circ \chi_{\mathrm{temp}}, \chi^*)$-fitting resize function $\omega^*$ for $V \setminus M$ and $f^* \circ \chi_{\mathrm{temp}}$ may need a domain of size $\Theta(n)$ since there might be $\Theta(n)$ bag colors. Hence, the number of $(f^* \circ \chi_{\mathrm{temp}}, \chi^*)$-fitting resize function $\omega^*$ for $V \setminus M$ and $f^* \circ \chi_{\mathrm{temp}}$ may exceed our desired running time. In other words, we cannot iterate over all $(f^* \circ \chi_{\mathrm{temp}}, \chi^*)$-fitting resize function $\omega^*$ for $V \setminus M$ and $f^* \circ \chi_{\mathrm{temp}}$ in the desired running time.

Hence, to obtain our desired running time, we want to iterate only over all possible functions $\omega\colon \mathrm{col}_M \to \mathbb{Z}$ such that there is a coloring $\chi^*$ of $V$ for which there is a template recolor-function $f^*$ and an $(f^*\circ\chi_{\mathrm{temp}}, \chi^*)$-fitting resize function $\omega^*$ for $V\setminus M$ and $f^*\circ\chi_{\mathrm{temp}}$ that fulfill $\omega = \omega^*\circ f^*|_{\mathrm{col}_M}$. That is, we only want to know how the colors of $\mathrm{col}_M$ are resized. For each such function $\omega$, we then compute the best such coloring $\chi^*$. Since checking whether such a function $\omega$ fulfills the desired properties is not trivial, we instead iterate over the set of functions

$$\mathcal{W}_X := \{\omega\colon X \to \mathbb{Z} \mid \sum_{i\in X} |\omega(i)| \leq 2k\}$$

for $X = \mathrm{col}_M$. Note that $\mathcal{W}_{\mathrm{col}_M}$ contains all functions $\omega$ described above, since each of them is a restriction of a resize function and, thus, fulfills $\sum_{i\in\mathrm{col}_M} |\omega(i)| \leq 2k$. Using a stars-and-bars type argument, one can show that $\mathcal{W}_{\mathrm{col}_M}$ can be computed in the desired running time [89, Observation 14].

**Claim 11.** Let $X$ be a finite set. The collection $\mathcal{W}_X := \{\omega\colon X \to \mathbb{Z} \mid \sum_{i\in X} |\omega(i)| \leq 2k\}$ can be computed in $2^{|X|}\cdot 3^{2k}\cdot(|X|+k)^{\mathcal{O}(1)}$ time.

Hence, it remains to show how to compute, for a function $\omega \in \mathcal{W}_{\mathrm{col}_M}$, the minimum number of required edge-modifications of any coloring $\chi^*$ such that there is a template recolor-function $f^*$ and an $(f^*\circ\chi_{\mathrm{temp}}, \chi^*)$-fitting resize function $\omega^*$ for $V\setminus M$ and $f^*\circ\chi_{\mathrm{temp}}$ that fulfill $\omega = \omega^*\circ f^*|_{\mathrm{col}_M}$, or to verify that no such functions exist. To this end, we describe a dynamic program.

**The dynamic program if the template recolor-function is known.** To develop a better understanding of the algorithm for the general task, we first show how to solve this task if the template recolor-function $f^*$ is known. Let $\mathrm{col}_M^* := \{f^*(\alpha) \mid \alpha \in \mathrm{col}_M\} = \mathrm{col}_{\mathrm{Mod}} \cup \{f^*(\alpha) \mid \alpha \in \mathrm{col}_{\mathrm{Move}}\}$ denote the set of colors that are assigned to modulator vertices by $\chi_{\mathcal{C}}$ or $f^*\circ\chi_{\mathrm{temp}}$. Thus, the task is to find a best coloring $\chi^*$ of $V$ such that

- $\chi_{\mathrm{int}} := f^*\circ\chi_{\mathrm{temp}}$ is the intermediate coloring for $\chi^*$ and
- there is a $(\chi_{\mathrm{int}}, \chi^*)$-fitting resize function $\omega^*$ for $V \setminus M$ and $\chi_{\mathrm{int}}$ that fulfills $\omega^*\circ f^*|_{\mathrm{col}_M^*} \in \mathcal{W}_{\mathrm{col}_M^*}$.

The dynamic programming table $D$ has entries of type $D[i,\omega]$ with $i \in [0, |\mathcal{B}|]$ and $\omega \in \mathcal{W}_{\mathrm{col}_M^*}$.

For each $i \in [0, |\mathcal{B}|]$, let $V_{\leq i}$ denote the union of the first $i$ bags. Recall that due to Observation 5.22 and Observation 5.23, for each bag $B_i \in \mathcal{B}$, there is at most one

bag color $\alpha_i$ with $\chi_{\mathcal{C}}^{-1}(\alpha_i) \subseteq B_i$. If such a bag color $\alpha_i$ exists for $B_i$, then we assume for simplicity that $\alpha_i = i$ and, otherwise, we assume that $i$ is not used by $\chi_{\text{temp}}$.

Each entry $D[i, \omega]$ stores the minimum cost of any coloring $\chi^*$ of $V_{\leq i} \cup M$, such that

- $\chi_{\text{int}}|_{V_{\leq i} \cup M}$ and $\chi^*$ agree on all vertices of $M$ and
- there is a $(\chi_{\text{int}}|_{V_{\leq i} \cup M}, \chi^*)$-fitting resize function[1] $\omega^*_{\leq i}$ for $V_{\leq i}$ and $\chi_{\text{int}}|_{V_{\leq i} \cup M}$ that fulfills $\omega = \omega^*_{\leq i}\big|_{\text{col}^*_{\text{M}}}$.

For the base case $i = 0$ we set

$$D[0, \omega] := \begin{cases} |(E \oplus \text{Cl}(\mathcal{C}_{\chi_{\text{int}}})) \cap \binom{M}{2}| & \text{if } \omega(\alpha) = 0 \text{ for each color } \alpha \in \text{col}^*_{\text{M}}, \text{ and} \\ \infty & \text{otherwise.} \end{cases}$$

This correct since $\chi_{\text{int}}$ and $\chi^*$ agree on all vertices of $M$. If there is a color $\alpha \in \text{col}^*_{\text{M}}$ with $\omega(\alpha) \neq 0$, setting the entry to $\infty$ is correct since $\omega$ is required to be the restriction of a resize function for $V_{\leq 0} = \emptyset$ and $\chi_{\text{int}}$, and for each resize function $\omega^*_{\leq 0}$ for $V_{\leq 0} = \emptyset$ and $f^* \circ \chi_{\text{temp}}$, and each color $\alpha \in \mathbb{N}$, we have $\omega^*_{\leq 0}(\alpha) = 0$.

To compute an entry of the dynamic programming table for $i > 0$, we have to consider four types of edge modifications:

(a) Those having both endpoints in $V_{\leq i-1} \cup M$,

(b) those having one endpoint in $B_i$ and the other endpoint in $M$,

(c) those having both endpoints in $B_i$, and

(d) those having one endpoint in $B_i$ and the other endpoint in $V_{\leq i-1}$.

To provide the recurrence for the dynamic programming table, we observe that we can assume that for each bag $B_i \in \mathcal{B}$, $\chi^*$ uses at most one color $\text{aux}_i$ distinct from $i$ with $\chi^{*-1}(\text{aux}_i) \subseteq B_i$. Essentially, this follows by the same observations that lead to Observations 5.22 and 5.23: (i) it is never optimal to introduce a new color $\alpha$ which is only assigned to bag vertices and vertices of different bags receive that color and (ii) it is never optimal to introduce more than one new color per bag $B \in \mathcal{B}$ that is assigned to only vertices of $B$.

[1] This is a slight abuse of notation, since resize functions are formally only defined for colorings of $V$.

**Observation 5.27.** *Let $\chi^*$ be a best coloring of $V$ for which $f^* \circ \chi_{\text{temp}}$ is the intermediate coloring such that there is an $(f^* \circ \chi_{\text{temp}}, \chi^*)$-fitting resize function $\omega^*$ for $V \setminus M$ and $f^* \circ \chi_{\text{temp}}$. Then, for each $i \in [1, |\mathcal{B}|]$, there is at most one color $\text{aux}_i \in \mathbb{N} \setminus (\text{col}_{\text{M}}^* \cup \text{col}_{\text{Move}} \cup \text{col}_{\text{Bag}})$ with $\chi^{*-1}(\text{aux}_i) \cap B_i \neq \emptyset$. Moreover, if such a color $\text{aux}_i$ exists, then $\chi^{*-1}(\text{aux}_i) \subseteq B_i$.*

We now set

$$D[i, \omega] := \min_{\substack{\omega_i \colon \text{col}_{\text{M}}^* \cup \{i,\ \text{aux}_i\} \to \mathbb{Z} \\ \omega_i \text{ is a resize function for } B_i \text{ and } \chi_{\text{int}}}} \Bigg\{ D[i-1, \omega - \omega_i] + \text{req}_{B_i}^{\chi_{\text{int}}, \omega_i} + \sum_{\alpha \in \text{col}_{\text{M}}^*} (|B_i \cap \chi_{\text{int}}^{-1}(\alpha)| + \omega_i(\alpha)) \cdot (|V_{\leq i-1} \cap \chi_{\text{int}}^{-1}(\alpha)| + \omega(\alpha) - \omega_i(\alpha)) \Bigg\}. \tag{5.3}$$

Here, $\text{req}_{B_i}^{\chi_{\text{int}}, \omega_i}$ denotes the minimum number of edge-modifications of Type (b) and Type (c) for the resize function $\omega_i$. Due to Claim 10, this value can be computed in polynomial time. Further, in the recurrence, $D[i-1, \omega - \omega_i]$ stores the minimum number of edge-modifications of Type (a).[2] Finally, the last line of Equation (5.3) denotes the required number of edge-modifications of Type (d). Since no two vertices in different bags are adjacent and since we are looking for a coloring $\chi^*$ where for each color $\alpha \in \text{col}_{\text{M}}^*$, $\omega_i(\alpha)$ additional vertices of $B_i$ receive color $\alpha$ and $\omega(\alpha) - \omega_i(\alpha)$ additional vertices of $V_{\leq i-1}$ receive color $\alpha$, we have to insert all $(|B_i \cap \chi_{\text{int}}^{-1}(\alpha)| + \omega_i(\alpha)) \cdot (|V_{\leq i-1} \cap \chi_{\text{int}}^{-1}(\alpha)| + \omega(\alpha) - \omega_i(\alpha))$ edges between these vertices.

Finally, the minimum number of edge-modifications of any sought coloring is stored in $\max_{\omega \in \mathcal{W}_{\text{col}_{\text{M}}^*}} D[|\mathcal{B}|, \omega]$. Moreover, a corresponding coloring can be computed via traceback.

**The actual dynamic program (where the template recolor-function is unknown).** Based on the dynamic program for $D$ (the case where the template recolor-function is known), we will now describe the actual dynamic program that finds the best coloring $\chi^*$ of $V$ for which there is a template recolor-function $f^*$ such that $f^* \circ \chi_{\text{temp}}$ is the intermediate coloring for $\chi^*$ and for which there is an $(f^* \circ \chi_{\text{temp}}, \chi^*)$-fitting resize function $\omega^*$ fulfilling $\omega^* \circ f^*|_{\text{col}_{\text{M}}} \in \mathcal{W}_{\text{col}_{\text{M}}}$.

[2] Recall that the domain of $\omega - \omega_i$ is $\text{col}_{\text{M}}^*$. Still, $\omega - \omega_i$ might not be a function of $\mathcal{W}_{\text{col}_{\text{M}}^*}$. In this case, $D[i-1, \omega - \omega_i]$ is not an entry of the dynamic programming table and we evaluate $D[i-1, \omega - \omega_i]$ as $\infty$.

In contrast to the previous dynamic program, we encounter the more complicated task of additionally finding the template recolor-function $f^*$. Hence, we also have to track the colors $S$ of $\mathrm{col}_{\mathrm{Move}}$ that may still be recolored to some bag color by $f^*$ while additionally ensuring that $f^*|_{\mathrm{col}_{\mathrm{Move}}}$ is injective. To this end, in contrast to the previous dynamic programming table $D$, our new dynamic programming table $T$ has an additional dimension that keeps track of the moving colors $S \subseteq \mathrm{col}_{\mathrm{Move}}$ that may be recolored to some bag color of $[1, i]$ by $f^*$. Moreover, to calculate all edge-modifications of Type (d) correctly while the value of $f^*(\alpha)$ is not yet determined for moving colors $\alpha \in S$, the dynamic programming table $T$ has an additional dimension storing some cost function $c\colon \mathrm{col}_{\mathrm{M}} \to \mathbb{Z}$. Intuitively, $c(\alpha)$ stores the number of vertices of $(V \setminus M) \setminus V_{\leq i}$, that is, the number of vertices of the last $|\mathcal{B}| - i$ bags, that were assigned the same color as $\alpha$ under $f^*$. Hence, when fixing the value of $f^*(\alpha)$ for some color $\alpha \in S$, we can account for the edge-modifications of Type (d) between vertices of $\chi^{*-1}(f^*(\alpha))$ and vertices of $(V \setminus M) \setminus V_{\leq i}$—that we were not able to account for earlier—by charging additional costs of $c(\alpha) \cdot \chi^{*-1}(f^*(\alpha))$. Note that by the above intuition, $c$ is the restriction of a resize function for $(V \setminus M) \setminus V_{\leq i}$ and $f^* \circ \chi_{\mathrm{temp}}$, which implies that it suffices to consider functions $c$ that fulfill $c \in \mathcal{W}_{\mathrm{col}_{\mathrm{M}}}$.

Each entry $T[i, S, \omega, c]$ stores the minimum value of $\mathrm{cost}(\mathcal{C}_{\chi^*}) + \sum_{\alpha \in S} c(\alpha) \cdot \chi^{*-1}(f^*(\alpha))$ of any coloring $\chi^*$ of $V_{\leq i} \cup M$ for which there is a template recolor-function $f^*$ such that

- there is an $(f^* \circ \chi_{\mathrm{temp}}, \chi^*)$-fitting resize function $\omega^*_{\leq i}$ for $V_{\leq i}$ and $f^* \circ \chi_{\mathrm{temp}}$ that fulfills $\omega = \omega^*_{\leq i} \circ f^*\big|_{\mathrm{col}_{\mathrm{M}}}$,
- colors of $S$ may be recolored to bag colors of $[1, i]$, that is, for each color $\alpha \in S$, $f^*(\alpha) \in \{\alpha\} \cup [1, i]$, and
- no color of $\mathrm{col}_{\mathrm{Move}} \setminus S$ is recolored to a bag color of $[1, i]$, that is, for each color $\alpha \in \mathrm{col}_{\mathrm{Move}} \setminus S$, $f^*(\alpha) \notin [1, i]$.

For the base case $i = 0$, analogously to the base case of $D$, we set for each $S \subseteq \mathrm{col}_{\mathrm{Move}}$ and each $c \in \mathcal{W}_{\mathrm{col}_{\mathrm{M}}}$

$$T[0, S, \omega, c] := \begin{cases} |(E \oplus \mathrm{Cl}(\mathcal{C}_{\chi_{\mathrm{temp}}})) \cap \binom{M}{2}| & \text{if } \omega(\alpha) = 0 \text{ for each } \alpha \in \mathrm{col}_{\mathrm{M}}, \text{ and} \\ \infty & \text{otherwise.} \end{cases}$$

Again, for an entry of the dynamic programming table for $i > 0$, we have to consider the four types of edge-modifications (a)–(d).

To compute an entry $T[i, S, \omega, c]$ for some $i > 0$, we consider two cases. In the first case, no color of $S$ is recolored to color $i$ under $f^*$ and we follow the recurrence

for $D$. We set

$$T^{\mathrm{id}}[i,S,\omega,c] := \min_{\substack{\omega_i\colon \mathrm{col}_{\mathrm{M}}\cup\{i,\ \mathrm{aux}_i\}\to\mathbb{Z} \\ \omega_i \text{ is a resize function for } B_i \text{ and } \chi_{\mathrm{temp}}}} \Bigg\{ T[i-1,S,\omega-\omega_i,c+\omega_i] + \mathrm{req}_{B_i}^{\chi_{\mathrm{temp}},\omega_i} + \sum_{\alpha\in\mathrm{col}_{\mathrm{M}}} (|B_i\cap\chi_{\mathrm{temp}}^{-1}(\alpha)|+\omega_i(\alpha))\cdot(|V_{\leq i-1}\cap\chi_{\mathrm{temp}}^{-1}(\alpha)|+\omega(\alpha)-\omega_i(\alpha)) \Bigg\}. \tag{5.4}$$

Here, again, $\mathrm{req}_{B_i}^{\chi_{\mathrm{temp}},\omega_i}$ denotes the minimum number of edge-modifications of Type (b) and Type (c) for the resize function $\omega_i$. Intuitively, the last line of Equation (5.4) resembles the required number of edge-modifications of Type (d). Note that the last line of Equation (5.4) reads as follows for colors $\alpha \in \mathrm{col}_{\mathrm{Move}}$: $\omega_i(\alpha)\cdot(\omega(\alpha)-\omega_i(\alpha))$. Hence, unfortunately, in contrast to the recurrence for $D$, this line does not necessarily cover all edge-modifications of Type (d): since we do not know the function $f^*$, for each moving color $\alpha \in S$, we cannot account for all edge-modifications incident with vertices of $V_{\leq i-1}$ that receive color $f^*(\alpha)$ under $f^*\circ\chi_{\mathrm{temp}}$. Hence, we have to account for all these edge-modifications, when we determine $f^*(\alpha)$ for a given color $\alpha \in S$. This is done by increasing for each color $\alpha \in \mathrm{col}_{\mathrm{M}}$ the value of the additional costs of $\alpha$ according to the cost function by $\omega_i(\alpha)$. Hence, $T[i-1,S,\omega-\omega_i,c+\omega_i]$ stores the minimum number of edge-modifications of Type (a) plus

$$\sum_{\alpha\in S} c(\alpha)\cdot\chi^{*-1}(f^*(\alpha)) + \sum_{\alpha\in S}\omega_i(\alpha)\cdot\chi^{*-1}(f^*(\alpha)).$$

Here, the latter sum represents the number of edge-modifications with one endpoint in $B_i$ and the other endpoint in $\chi^{*-1}(f^*(\alpha))$. In other words, $T[i-1,S,\omega-\omega_i,c+\omega_i]$ and the last line of Equation (5.4) account for the minimum number of edge-modifications of Type (a) and Type (d) plus the additional cost of $\sum_{\alpha\in S} c(\alpha)\cdot\chi^{*-1}(f^*(\alpha))$. Similar to the computation of the dynamic programming table $D$, if $\omega-\omega_i \notin \mathcal{W}_{\mathrm{col}_{\mathrm{M}}}$ or $c+\omega_i \notin \mathcal{W}_{\mathrm{col}_{\mathrm{M}}}$, then $T[i-1,S,\omega-\omega_i,c+\omega_i]$ is not an entry of the dynamic programming table and we instead evaluate $T[i-1,S,\omega-\omega_i,c+\omega_i]$ as $\infty$.

Next, we present the recurrence for the case, where some color of $S$ receives color $i$

under $f^*$. For the second case, we now set

$$T^{\text{move}}[i,S,\omega,c] := \min_{\beta\in S} \min_{\substack{\omega_i\colon \text{col}_\text{M}\cup\{\text{aux}_i\}\to\mathbb{Z} \\ \omega_i\circ\text{id}_{[i\mapsto\beta]} \text{ is a resize function for } B_i \text{ and } \text{id}_{[\beta\mapsto i]}\circ\chi_\text{temp}}} \Bigg\{ T[i-1, S\setminus\{\beta\}, \omega-\omega_i, c+\omega_i] + \text{req}_{B_i}^{\text{id}_{[\beta\mapsto i]}\circ\chi_\text{temp},\,\omega_i\circ\text{id}_{[i\mapsto\beta]}} + \sum_{\alpha\in\text{col}_\text{M}} (|B_i\cap\chi_\text{temp}^{-1}(\alpha)|+\omega_i(\alpha))\cdot(|V_{\le i-1}\cap\chi_\text{temp}^{-1}(\alpha)|+\omega(\alpha)-\omega_i(\alpha)) + |\chi_\text{temp}^{-1}(i)|\cdot c(\beta)\Bigg\} \tag{5.5}$$

Note that this recurrence is nearly identical to the one for $T^{\text{id}}[i,S,\omega,c]$ except for two main aspects: First, we take the best color $\beta\in S$ to be recolored to $i$ by $f^*$. Second, in the last line of Equation (5.5), we account for the additional costs charged by the cost function $c$, since we now determine the value $f^*(\beta)$.[3]

To combine these two cases, we set

$$T[i,S,\omega,c] := \begin{cases} T^{\text{id}}[i,S,\omega,c] & i\notin\text{col}_\text{Bag}\vee S=\emptyset \text{ and} \\ \min(T^{\text{id}}[i,S,\omega,c], T^{\text{move}}[i,S,\omega,c]) & \text{otherwise.}\end{cases}$$

Finally, the minimum number of edge-modifications of any sought coloring is exactly $\min_{\omega\in\mathcal{W}_{\text{col}_\text{M}}} T[|\mathcal{B}|,\text{col}_\text{Move},\omega,\mathbf{0}]$, where $\mathbf{0}$ is the function of $\mathcal{W}_{\text{col}_\text{M}}$ that assigns value 0 to each color of $\text{col}_\text{M}$. Moreover, a corresponding coloring can be computed via traceback. Hence, if there is a coloring $\chi'$ of $V$ with $d_\text{move}(\chi_\mathcal{C},\chi')\le k$ that improves over $\chi_\mathcal{C}$ where $\chi_\text{temp}$ is quasi-intermediate for $\chi'$, then, for some function $\omega\in\mathcal{W}_{\text{col}_\text{M}}$, $T[|\mathcal{B}|,\text{col}_\text{Move},\omega,\mathbf{0}]<\text{cost}(\chi_\mathcal{C})$.

It remains to prove that the presented algorithm has the stated running time. The table $T$ has $|\mathcal{B}|\cdot 2^{|\text{col}_\text{Move}|}\cdot|\mathcal{W}_{\text{col}_\text{M}}|\cdot|\mathcal{W}_{\text{col}_\text{M}}| \le 2^{|\text{col}_\text{Move}|}\cdot 4^{|\text{col}_\text{Mod}|+|\text{col}_\text{Move}|}\cdot 9^{2k}\cdot n^{\mathcal{O}(1)}$ entries. Each such entry $T[i,S,\omega,c]$ can be computed in $|\mathcal{W}_{\text{col}_\text{M}\cup\{i,\text{aux}_i\}}|\cdot n^{\mathcal{O}(1)} \le 2^{|\text{col}_\text{Mod}|+|\text{col}_\text{Move}|}\cdot 3^{2k}\cdot n^{\mathcal{O}(1)}$ time, since each resize function for $B_i$ is contained in $\mathcal{W}_{\text{col}_\text{M}\cup\{i,\text{aux}_i\}}$. Hence the total running time evaluates to $2^{\mathcal{O}(|\text{col}_\text{Mod}|+|\text{col}_\text{Move}|+k)}\cdot n^{\mathcal{O}(1)}$ time. Recall that $\chi_\text{temp}$ is a template coloring, so $d_\text{move}(\chi_\mathcal{C},\chi_\text{temp})\le k$ and, thus, $|\text{col}_\text{Move}|\le k$. Consequently, we obtain the stated running time of $2^{\mathcal{O}(|\text{col}_\text{Mod}|+k)}\cdot n^{\mathcal{O}(1)}$ time. □

We now conclude our permissive algorithm for LS Cluster Editing. That is, we now prove Theorem 5.21. As we can switch between clusterings and cluster colorings in polynomial time, the following statement is equivalent to Theorem 5.21.

[3]Again, if $\omega-\omega_i\notin\mathcal{W}_{\text{col}_\text{M}}$ or $c+\omega_i\notin\mathcal{W}_{\text{col}_\text{M}}$, $T[i-1,S,\omega-\omega_i,c+\omega_i]$ is not an entry of the dynamic programming table and we instead evaluate $T[i-1,S,\omega-\omega_i,c+\omega_i]$ as $\infty$.

**Theorem 5.28.** *Let $G = (V, E)$ be a graph, let $\chi_{\mathcal{C}}$ be a coloring of $V$, and let $k \in \mathbb{N}$. In $2^{\mathcal{O}(k)} \cdot k^k \cdot \mathrm{cvd}^{3k} \cdot n^{\mathcal{O}(1)}$ time, one can find a coloring $\chi^*$ that improves over $\chi_{\mathcal{C}}$ or correctly output that there is no coloring in the $k$-move neighborhood of $\chi_{\mathcal{C}}$ that improves over $\chi_{\mathcal{C}}$.*

*Proof.* First, we compute a 2-approximate cluster modulator $M$ of $G$ in polynomial time [8]. Let $\mathcal{B}$ be the collection of bags of $G[V \setminus M]$, let $\mathrm{col}_{\mathrm{Mod}}$ and $\mathrm{col}_{\mathrm{Bag}}$ be the modulator colors and bag colors of $\chi_{\mathcal{C}}$, respectively. If the condition of Observation 5.22 or Observation 5.23 applies, then we find a coloring $\chi^*$ of $V$ that improves over $\chi_{\mathcal{C}}$ in polynomial time. Hence, assume in the following that this is not the case.

As mentioned at the beginning of this section, we perform an initial branching step to ensure that the coloring $\chi_{\mathcal{C}}$ uses at most $2k$ modulator colors. Let $\chi'$ be the best coloring in the $k$-move neighborhood of $\chi_{\mathcal{C}}$. Assume that $\chi'$ improves over $\chi_{\mathcal{C}}$. Since $d_{\mathrm{move}}(\chi_{\mathcal{C}}, \chi') \leq k$, Observation 5.2 implies that there is a subset $S \subseteq \mathrm{col}_{\mathrm{Mod}}$ of size at least $|\mathrm{col}_{\mathrm{Mod}}| - 2k$ such that for each modulator color $\alpha \in S$, $\chi_{\mathcal{C}}^{-1}(\alpha) = \chi'^{-1}(\alpha)$. Hence, to find a coloring that improves over $\chi_{\mathcal{C}}$ it is sufficient to branch into all subsets $S \subseteq \mathrm{col}_{\mathrm{Mod}}$ of size at least $|\mathrm{col}_{\mathrm{Mod}}| - 2k$ and ask for a coloring $\widehat{\chi}'$ of $\widehat{V} := V \setminus (\cup_{\alpha \in S} \chi_{\mathcal{C}}^{-1}(\alpha))$ that improves over $\widehat{\chi} := \chi_{\mathcal{C}}|_{\widehat{V}}$ with respect to the subgraph $G[\widehat{V}]$. Note that this branching takes $|\mathrm{col}_{\mathrm{Mod}}|^{2k} \cdot n^{\mathcal{O}(1)} \leq |M|^{2k} \cdot n^{\mathcal{O}(1)}$ time and in each branching case, the corresponding coloring has at most $2k$ modulator colors.

Hence, in the following, we assume that $\mathrm{col}_{\mathrm{Mod}}$ has size at most $2k$. Next, we iterate over a maximal set $\mathcal{X}$ of pairwise non-isomorphic template colorings and compute for each such template coloring $\chi_{\mathrm{temp}}$ a coloring $\chi^*$ which is at least as good as a best coloring $\chi''$ in the $k$-move neighborhood of $\chi_{\mathcal{C}}$ where $\chi_{\mathrm{temp}}$ is quasi-intermediate for $\chi''$. The latter task can be done in $2^{\mathcal{O}(|\mathrm{col}_{\mathrm{Mod}}|+k)} \cdot n^{\mathcal{O}(1)}$ time due to Lemma 5.26 for each template coloring $\chi_{\mathrm{temp}} \in \mathcal{X}$. Due to Lemma 5.25, $\mathcal{X}$ can be computed in $(|\mathrm{col}_{\mathrm{Mod}}|+k)^k \cdot |M|^k \cdot n^{\mathcal{O}(1)}$ time and has size $(|\mathrm{col}_{\mathrm{Mod}}|+k)^k \cdot |M|^k \cdot n^{\mathcal{O}(1)}$.

This algorithm is correct, since there is a template coloring $\chi'_{\mathrm{temp}}$ such that $\chi'_{\mathrm{temp}}$ is quasi-intermediate for $\chi'$. Thus, in this way, we will find a coloring at least as good as $\chi'$.

The whole algorithm runs in $|M|^{2k} \cdot (|\mathrm{col}_{\mathrm{Mod}}|+k)^k \cdot |M|^k \cdot 2^{\mathcal{O}(|\mathrm{col}_{\mathrm{Mod}}|+k)} \cdot n^{\mathcal{O}(1)}$ time. Since we ensured with the initial branching, that $\mathrm{col}_{\mathrm{Mod}}$ has size at most $2k$, this results in a running time of $2^{\mathcal{O}(k)} \cdot k^k \cdot |M|^{3k} \cdot n^{\mathcal{O}(1)}$ time. Finally, since $M$ is a 2-approximate cluster modulator, $|M| \leq 2 \cdot \mathrm{cvd}(G)$. Hence, we obtain the stated running time of $2^{\mathcal{O}(k)} \cdot k^k \cdot \mathrm{cvd}(G)^{3k} \cdot n^{\mathcal{O}(1)}$ time. □

## 5.5 Concluding Remarks

In this chapter, we analyzed the parameterized complexity for LS Cluster Editing and LS Cluster Deletion. We showed that both problems are W[1]-hard when parameterized by $k$ plus the largest cluster of the initial clustering (see Section 5.3 and Theorem 5.14). Moreover, we showed—similar to the previous chapters—that both problems cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function $f$, unless the ETH fails. This running time lower bound also holds for the permissive version of LS Cluster Deletion. Positively, we present algorithms for both problems that run in $(3 \cdot e)^k \cdot \Delta^{2k} \cdot n^{\mathcal{O}(1)}$ time (see Theorem 5.6). Additionally, we presented algorithms for the permissive versions of both problems that run in $(\mathrm{cvd} \cdot k)^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time, where cvd denotes the cluster vertex deletion number of the input graph (see Theorems 5.20 and 5.28).

**Open questions.** From a theoretical point, we leave two questions open with respect to the considered problems. First, does LS Cluster Deletion admit an FPT-algorithm with respect to $k$ if the initial clustering contains only a constant number $|\mathcal{C}|$ of clusters? If this is the case, one might look into the combined parameter $k + |\mathcal{C}|$ for LS Cluster Deletion. Note that this parameter combination is not of interest for LS Cluster Editing, since we showed that LS Cluster Editing is W[1]-hard when parameterized by $k$ even if the initial clustering contains only two clusters. Second, can we show lower bounds for the permissive variant of LS Cluster Editing? So far, all of our negative results only hold for the strict version of LS Cluster Editing. In particular, the question is open, whether the permissive version of LS Cluster Editing admits the same $n^{\Omega(k)}$ running time lower bound as the permissive local search problems considered in Chapters 3 and 4.

It is also possible to think of further ways to improve upon our results. For example, again the question arises whether we can solve the considered problems in $h^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. Alternatively, one could consider other parameters smaller than the maximum degree. For example, can we solve LS Cluster Editing and LS Cluster Deletion in $\ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time for $\ell$ being the maximum number of edge deletions incident with any vertex with respect to the initial clustering?

Another direction is to generalize our results to related or more general problems. For example, one could consider weighted versions of LS Cluster Editing and LS Cluster Deletion. Here, we are given an additional weight function $\omega\colon \binom{V}{2} \to \mathbb{R}$. This weight function assigns a positive weight to each edge of the input graph and a negative weight to each non-edge, and the goal is to find a clustering $\mathcal{C}$ that maximizes the total weight of $\mathrm{Cl}(\mathcal{C})$. Alternatively, one can look into the complexity

of local search versions of related problems like Modularity Clustering [25,130] or $k$-Means Clustering [45]. (Note that the $k$ in $k$-Means Clustering is not the search radius but the maximum number of clusters in any valid solution to this problem.)

Finally, it would be interesting to see if our algorithmic ideas can be used to improve the local-search-based heuristics for Cluster Deletion or Cluster Editing. In particular, it would be interesting to analyze whether an implementation of the FPT-algorithms for $\Delta + k$ (see Theorem 5.6) can be used as a successful post-processing step for state-of-the-art heuristics.

# Chapter 6

# On the Complexity of Parameterized Local Search for the Maximum Parsimony Problem

Maximum Parsimony is one of the most popular methods for inferring phylogenetic (evolutionary) trees from sequences of morphological or molecular characters (for example the base pairs of DNA sequences) [54]. A common strategy to attack this notoriously hard problem is to perform a local search over the phylogenetic tree space. In this chapter, we analyze the complexity of local search for Maximum Parsimony with respect to several natural scalable local neighborhoods. Given sequences of characters for $n$ taxa, Maximum Parsimony aims to reconstruct a phylogenetic tree $T$ whose $n$ leaves are labeled bijectively by the $n$ taxa and that has the minimum *parsimony score* over all such trees. The parsimony score is the number of character state changes along the tree edges that are necessary when extending the sequences for the leaves of $T$ to all internal vertices of $T$. An example of the parsimony score of a fixed tree is depicted in Figure 6.1. Note that for each character $c$, this score is at least $s_c - 1$, where $s_c$ denotes the number of different character states. A phylogenetic tree is called *perfect* if it achieves score $s_c - 1$ for each character $c$. Such a perfect phylogeny does not always exists. For any given binary tree, the parsimony score can be computed in polynomial time using Fitch's algorithm [58] or Sankoff's algorithm [148].

From an algorithmic point of view, the Maximum Parsimony problem is notoriously hard: It is NP-complete even for binary characters [61]. Moreover, the current best running time is $\Omega((2n-3)!!)$ [28], where !! denotes the double factorial. That is, $(2n-3)!! = 1 \cdot 3 \cdot \ldots \cdot (2n-5) \cdot (2n-3)$. The associated algorithm generates

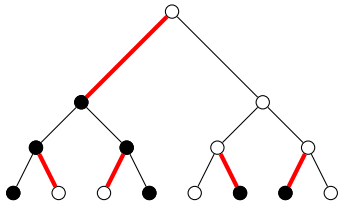


**Figure 6.1:** An example of the parsimony score of a given tree. The leaves of the trees are the taxa that are each assigned with a single character which can be either black or white. The parsimony score of the depicted tree is 5, since five edges have endpoints of different color.

all possible binary phylogenetic trees on $n$ leaves in a bottom-up fashion. Hence, the best known algorithm is essentially a brute-force-method. This running time bound is impractical when $n > 15$. Better running times are possible when the instance has a near-perfect phylogeny and the maximum number of different character states $s$ is small. Here, the running time is measured also in terms of the excess $q$ over the score of a perfect phylogeny. In the general case, Maximum Parsimony can be solved in $nm^{\mathcal{O}(q)}2^{\mathcal{O}(q^2s^2)}$ time [56], where $m$ is the length of the character sequences. Later, the running time was improved to $\mathcal{O}(21^q + 8^q nm^2)$ for the special case of binary characters and the practical usefulness of the improved algorithm was demonstrated for $q \leq 10$ [156]. In the worst case, however, $q$ can be essentially as large as $m$. Moreover, Maximum Parsimony is NP-hard even for $q = 0$ when the number of different character states is unbounded [22].

Given the above-described hardness of Maximum Parsimony, solving this problem exactly is currently impossible for many real-world datasets due to prohibitively high running times. Consequently, heuristic approaches, in particular hill-climbing local search algorithms, play an important role in computing good, but not necessarily optimal, solutions [6,63,64,71–74,132,143]. As the local neighborhood, one usually considers all trees that can be obtained by one or few rearrangement operations. The most well-known rearrangement operations on trees that are also considered in local search approaches for Maximum Parsimony [3] are nearest neighbor interchange (NNI), subtree prune and regraft (SPR), and tree bisection and reconnection (TBR) which are all formally defined in Section 6.1. Each of these operations deletes an edge of a tree and then reconnects the resulting two subtrees. Depending on the operation, the reconnection is more or less restrictive, with SPR being a generalization of NNI and TBR being a generalization of SPR. For such a rearrangement operation $\circ$, we can then define a distance between two trees $T$ and $T'$ with respect to $\circ$ as the minimum number of consecutive $\circ$-operations needed to transform $T$ into $T'$.

These distance measures then define scalable local neighborhoods for phylogenetic trees. The set of all trees that can be obtained by one operation is called the NNI, SPR, or TBR neighborhood, respectively. More generally, we say that a tree $T'$ is in the $k$-neighborhood with respect to NNI, SPR, or TBR of another tree $T$, if the distance from $T$ to $T'$ is at most $k$ with respect to NNI, SPR, or TBR operations, respectively.

In addition to NNI, SPR, and TBR, the $k$-ECR operation has also been considered in the literature (see for example the works by Ganapathy et al. [63, 64]). This latter operation first contracts up to $k$ edges and then refines the resulting tree arbitrarily. The $k$-ECR neighborhood contains all trees that can be obtained from a starting tree by applying one $k$-ECR operation. The 1-ECR neighborhood is exactly the NNI neighborhood, but the 2-ECR neighborhood strictly contains the set of trees reachable by two NNI moves [64]. The $k$-ECR neighborhood appeared earlier implicitly under the term *sectorial search* [74]. Moreover, it was observed that two binary phylogenetic trees are one $k$-ECR operation apart if and only if their Robinson-Foulds-distance is at most $2k$ [63]. A restricted version of the $k$-ECR neighborhood where the contracted edges must form a subtree (called the $k$-sECR neighborhood) was considered by Sankoff et al. [149]. They found that for larger values of $k$, the $k$-sECR neighborhood gives better results than the 1-ECR neighborhood or, equivalently, the NNI neighborhood. Guo et al. [85] found that exploring the $k$-ECR neighborhood is too costly and thus proposed a restriction of this neighborhood which already leads to very good local optima. Their approach contracts $k$ edges and then refines the resulting tree by using neighbor joining, a fast distance-based method to reconstruct phylogenetic trees. To summarize, local search is an important paradigm for designing heuristics for Maximum Parsimony, and it has been noted that larger neighborhoods, such as the $k$-ECR neighborhood, give better results at the cost of higher running times. So far, there is, however, no study on how hard exploring larger neighborhoods actually is.

As in the previous chapters, we again ask whether one can find algorithms that search the $k$-neighborhoods, with respect to the described distance measures, faster than $|I|^{\mathcal{O}(k)}$ time. Recall that a running time of $f(k) \cdot |I|^{\mathcal{O}(1)}$ would be desirable since the explosion in the running time would then depend only on $k$ and not on $|I|$.

**Our results.** In Section 6.1, we provide the problem-specific notation for this chapter. Afterwards, in Section 6.2, we provide new insights in the relation between the different considered distance measures. In Section 6.3, we show that even when all characters are binary, searching the $k$-ECR neighborhood is W[1]-hard with respect to $k$. The reduction that establishes this result also shows that, under the ETH,

a running time of $|I|^{\Omega(k)}$ is necessary. Moreover, the reduction implies hardness for searching the $k$-neighborhood with respect to NNI, SPR, and TBR. In a nutshell, our results show that one cannot gain a substantial speed-up over the brute-force algorithm when trying to search these large neighborhoods. In Section 6.4, we then lift the negative results to the permissive version of local search for all these distance measures. We then establish that $n^{\mathcal{O}(k)} \cdot m$ time is sufficient to search the $k$-neighborhoods with respect to any of NNI, SPR, TBR, and $k$-ECR, giving tight upper and lower bounds for running times that grow strongly with respect to $k$ only. Finally, we observe in Section 6.5 that the $k$-sECR neighborhood of Sankoff [149] can be searched in $k^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time, making it possible to consider much larger values of $k$ than for the other neighborhoods. Let us remark that, while we formally study the decision problem that asks for the existence of a better tree in the $k$-neighborhood, our hardness results and algorithms also apply to the problem of finding a best tree in the $k$-neighborhood.

## 6.1 Problem-Specific Notation

In this section, we formally define the concepts used in this chapter. This includes phylogenetic trees, the parsimony score, the considered distance measures on phylogenetic trees, and the local search problem we analyze with respect to these distance measures.

**Phylogenetic trees.** Throughout this chapter, $X$ denotes a non-empty finite set of *taxa*. An *(unrooted phylogenetic) $X$-tree* $T$ is a tree with leaf-set $X$ and where no vertex has degree 2. If all non-leaf vertices of $T$ have degree three, then $T$ is called *binary*. Furthermore, if an edge $e$ is incident with a leaf of $T$, then $e$ is called a *pendant edge* and, otherwise, an *internal edge*. For two disjoint sets of taxa $A$ and $B$, we say that $A|B$ is a *split* of an $X$-tree $T$ if there is an edge $e$ in $T$ such that the deletion of $e$ results in two subtrees where one has leaf set $A$ and the other has leaf set $B$. The set of all splits of $T$ is denoted by $\Sigma(T)$. Furthermore, we say that an $X$-tree $T'$ is a *refinement* of $T$ if $\Sigma(T) \subseteq \Sigma(T')$. Additionally, if $T'$ is binary, then $T'$ is a *binary refinement* of $T$. We say that two $X$-trees $T$ and $T'$ are *isomorphic* if $\Sigma(T) = \Sigma(T')$. Equivalently, two $X$-trees $T$ and $T'$ are *isomorphic* if there is a bijection $\varphi$ between the vertices of $T$ and the vertices of $T'$ such that $\varphi(x) = x$ for each taxon $x \in X$, and for all distinct vertices $u$ and $v$ of $T$, $\{u, v\}$ is an edge of $T$ if and only if $\{\varphi(u), \varphi(v)\}$ is an edge of $T'$.

Now, let $T$ be an $X$-tree and let $V'$ be a subset of the vertices of $T$. Then

$T(V')$ denotes the minimal subtree of $T$ containing all vertices in $V'$. Moreover, the *restriction* of $T$ to a subset of taxa $A \subseteq X$, denoted by $T|_A$, is the $A$-tree obtained from $T(A)$ by suppressing every degree-2 vertex. Let $A$ be a non-empty and proper subset of $X$ and let $T$ be a binary $X$-tree. If $A|(X \setminus A)$ is a split of $T$, then the subtree $T(A)$ is a *pendant $A$-tree*. Moreover, the *pseudo-root* of $T(A)$ is the unique vertex of degree 2 in $T(A)$ if $|A| > 1$ and the unique vertex of $T(A)$, otherwise.

**Maximum parsimony.** A *character*[1] $c$ on $X$ is a function $c\colon X \to C$. If $|C| = 2$, then $c$ is called a *binary* character. Intuitively, $C$ can be thought of as the underlying alphabet and each element in the alphabet is a *character state*. Let $T$ be an $X$-tree with vertex set $V$, and let $c$ be a character on $X$ whose set of character states is $C$. An *extension* $c^*$ of $c$ to $V$ is a function $c^*\colon V \to C$ such that $c^*(x) = c(x)$ for each taxon $x \in X$. Let $c^*$ be an extension of $c$. A *mutation edge of $c^*$ in $T$* is an edge $\{u, v\}$ in $T$ such that $c^*(u) \neq c^*(v)$ and we let $\mathrm{score}_{c^*}(T)$ denote the number of mutation edges of $c^*$ in $T$. Then the *parsimony score* of $c$ on $T$, denoted by $\mathrm{score}_c(T)$, is obtained by minimizing $\mathrm{score}_{c^*}(T)$ over all possible extensions $c^*$ of $c$. An extension $c^*$ that minimizes $\mathrm{score}_{c^*}(T)$ is called an *optimal extension of $c$ in $T$*. Moreover the *maximum parsimony score* of $c$, denoted by $\mathrm{MP}(c)$, is the parsimony score of $c$ minimized over all binary $X$-trees.

Now let $S = (c_1, c_2, \ldots, c_m)$ be a sequence of characters on $X$. Then the parsimony score of $S$ on an $X$-tree $T$ is defined as $\mathrm{score}_S(T) = \sum_{i=1}^{m} \mathrm{score}_{c_i}(T)$ and, similarly, the maximum parsimony score of $S$, denoted by $\mathrm{MP}(S)$, is the parsimony score of $S$ minimized over all binary $X$-trees.

We may abuse notation by writing $c \in S$ if the character $c$ is contained in the sequence $S$.

**Observation 6.1.** *Let $T$ be a binary $X$-tree, let $c$ be a character, and let $c^*$ be an optimal extension of $c$ on $T$. If there is a split $A|B$ of $T$ such that $c(a) = c(a')$ for each pair of taxa $a$ and $a'$ of $A$, then $c^*(v) = c(a)$ for each vertex $v$ of $T(A)$.*

**SPR and TBR.** Let $T$ be a binary $X$-tree. Let $e = \{u, v\}$ be an edge of $T$, and let $T_1$ and $T_2$ be the two trees obtained from $T$ by deleting $e$ and suppressing $u$ if its degree is 2. Without loss of generality, we may assume that $T_2$ contains $v$. If $T_1$ contains at least one edge, subdivide an edge of $T_1$ with a new vertex $u'$; otherwise, set $u'$ to be the single isolated vertex of $T_1$. Finally, obtain a binary $X$-tree $T'$ by adding the new edge $\{u', v\}$. We say that $T'$ has been obtained from $T$ by a single

[1] Characters as defined here are not elements of some alphabet but functions that assign an element of some alphabet to each taxon.

*subtree prune and regraft (SPR)* operation. We next define a generalization of the SPR operation. Again, let $e$ be an edge of $T$, and let $T_1$ and $T_2$ be the two trees obtained from $T$ by deleting $e$ and suppressing any resulting degree-2 vertices. For each $i \in \{1, 2\}$, if $T_i$ has at least one edge, subdivide an edge in $T_i$ with a new vertex $v_i$ and, otherwise, set $v_i$ to be the single vertex of $T_i$. Obtain a binary $X$-tree $T'$ by adding the new edge $\{v_1, v_2\}$. We say that $T'$ has been obtained from $T$ by a single *tree bisection and reconnection (TBR)* operation.

**NNI, $k$-ECR, and $k$-sECR.** Let $T$ be a binary $X$-tree. Let $e = \{u, v\}$ be an edge of $T$ and let $e' = \{v, w\}$ be an internal edge of $T$ that is adjacent to $e$. Let $T'$ be a binary $X$-tree obtained from $T$ by deleting $e$, suppressing $v$, subdividing an edge that is incident with $w$ with a new vertex $v'$, and joining $u$ and $v'$ via a new edge. We say that $T'$ has been obtained from $T$ by a single *nearest neighbor interchange (NNI)* operation. Equivalently, if $T'$ is a binary refinement of the tree obtained from $T$ by contracting $e'$ and $T'$ is non-isomorphic to $T$, then $T'$ is obtained from $T$ by a single NNI operation.

Now let $T$ be a binary $X$-tree, and let $k$ be a positive integer. Let $T'$ be a binary refinement of a tree obtained from $T$ by contracting $k$ (distinct) internal edges $E'$. If $T'$ and $T$ are non-isomorphic, then we say that $T'$ is a single *$k$-edge contract and refine ($k$-ECR)* operation [63] apart from $T$ and that $E'$ is a *contraction set* for $T$ and $T'$. Note that an NNI operation is a 1-ECR operation and vice versa. We denote the restricted version of a $k$-ECR operation that requires the $k$ contracted edges to form a subtree of $T$ as $k$-sECR [149].

**Distance measures.** Let $T$ and $T'$ be binary $X$-trees. For each operation $\Theta \in \{\mathrm{NNI}, \mathrm{SPR}, \mathrm{TBR}\}$, the distance $d_\Theta(T, T')$ is defined as the minimum number of $\Theta$ operations to transform $T$ into $T'$ [3]. The distance $d_{\mathrm{ECR}}(T, T')$ is defined as the smallest number $k$ such that $T$ and $T'$ are one $k$-ECR operation apart. Analogously, the distance $d_{\mathrm{sECR}}(T, T')$ is defined as the smallest number $k$ such that $T$ and $T'$ are one $k$-sECR operation apart.

Another famous distance measure between $X$-trees is $d_{\mathrm{RF}}$, the *Robinson-Foulds-distance* [145]. This distance measure is define by $d_{\mathrm{RF}}(T, T') := |\Sigma(T) \oplus \Sigma(T')|$ for any two phylogenetic $X$-trees $T$ and $T'$. It was shown that for binary phylogenetic $X$-trees $T$ and $T'$, $d_{\mathrm{RF}}(T, T') = 2 \cdot d_{\mathrm{ECR}}(T, T')$ [63].

**Considered problems.** In this chapter, we consider for each distance measure $d \in \{d_{\mathrm{NNI}}, d_{\mathrm{SPR}}, d_{\mathrm{TBR}}, d_{\mathrm{ECR}}, d_{\mathrm{RF}}, d_{\mathrm{sECR}}\}$ the parameterized complexity of the following problem.

$d$-LS Maximum Parsimony
**Input:** A set of taxa $X$, a binary $X$-tree $T$, a sequence of characters $S$, and $k \in \mathbb{N}$.
**Question:** Does there exist a binary $X$-tree $T'$ with $d(T, T') \leq k$ and $\text{score}_S(T') < \text{score}_S(T)$?

## 6.2 Properties of the Considered Distance Measures

In this section, we analyze the relation of the different distance measures.

**Observation 6.2** ([3,144])**.** *The distance measures $d_{\text{NNI}}$, $d_{\text{SPR}}$, and $d_{\text{TBR}}$ are metrics.*

Since $d_{\text{RF}}$ is a metric [145] and for any two phylogenetic $X$-trees $T$ and $T'$, $d_{\text{RF}}(T, T') = 2 \cdot d_{\text{ECR}}(T, T')$, we conclude the following.

**Observation 6.3.** *The distance measure $d_{\text{ECR}}$ is a metric.*

Note that the distance measure $d_{\text{sECR}}$ is not a metric, since $d_{\text{sECR}}$ does not fulfill the triangle inequality.

For any two binary $X$-trees $T$ and $T'$, $d_{\text{NNI}}(T, T') \geq d_{\text{SPR}}(T, T') \geq d_{\text{TBR}}(T, T')$ [3]. In the following, we show that $d_{\text{NNI}}(T, T') \geq d_{\text{ECR}}(T, T')$ and that $d_{\text{ECR}}(T, T') \geq d_{\text{TBR}}(T, T')$. We start by showing that $d_{\text{NNI}}(T, T') \geq d_{\text{ECR}}(T, T')$.

**Lemma 6.4.** *Let $T$ and $T'$ be binary $X$-trees. Then, $d_{\text{NNI}}(T, T') \geq d_{\text{ECR}}(T, T')$.*

*Proof.* We show this statement by induction over $d_{\text{NNI}}(T, T')$. For the base case, consider $d_{\text{NNI}}(T, T') \leq 1$. Since $d_{\text{NNI}}(T, T') = 0$ implies that $T$ is equal to $T'$, $d_{\text{ECR}}(T, T') = 0$ if $d_{\text{NNI}}(T, T') = 0$. Moreover, since the NNI operations are exactly the 1-ECR operations, $d_{\text{ECR}}(T, T') = 1$ if $d_{\text{NNI}}(T, T') = 1$.

For the inductive step, suppose that $d_{\text{NNI}}(T, T') > 1$ and that the statement holds for each pair of binary $X$-trees $\widehat{T}$ and $\widetilde{T}$ with $d_{\text{NNI}}(\widehat{T}, \widetilde{T}) < d_{\text{NNI}}(T, T')$. Since $d_{\text{NNI}}(T, T') > 1$, by definition of $d_{\text{NNI}}$, there is an $X$-tree $\bar{T}$ with $d_{\text{NNI}}(T, \bar{T}) = d_{\text{NNI}}(T, T') - 1$ and $d_{\text{NNI}}(\bar{T}, T') = 1$. Hence, based on the induction hypothesis, $d_{\text{ECR}}(T, \bar{T}) \leq d_{\text{NNI}}(T, T') - 1$ and $d_{\text{ECR}}(\bar{T}, T') \leq 1$. Due to Observation 6.3, the triangle inequality implies $d_{\text{ECR}}(T, T') \leq d_{\text{ECR}}(T, \bar{T}) + d_{\text{ECR}}(\bar{T}, T') \leq d_{\text{NNI}}(T, T')$. □

Hence, to show that $d_{\text{NNI}}(T, T') \geq d_{\text{ECR}}(T, T') \geq d_{\text{SPR}}(T, T') \geq d_{\text{TBR}}(T, T')$ for any two binary $X$-trees $T$ and $T'$, it remains to show that $d_{\text{ECR}}(T, T') \geq d_{\text{SPR}}(T, T')$. To this end, we first observe the following connection between $d_{\text{sECR}}$ and $d_{\text{ECR}}$.

**Observation 6.5.** *Let $T$ and $T'$ be distinct binary $X$-trees and let $k > 0$ be an integer. If $d_{\mathrm{ECR}}(T, T') = k$, then there is a binary $X$-tree $\widetilde{T}$ with $d_{\mathrm{sECR}}(\widetilde{T}, T') > 0$ such that $d_{\mathrm{ECR}}(T, T') = d_{\mathrm{ECR}}(T, \widetilde{T}) + d_{\mathrm{sECR}}(\widetilde{T}, T')$.*

The idea behind Observation 6.5 is to consider the connected components of $T$ induced by the contraction set $S$ between $T$ and $T'$. If $S$ forms a subtree of $T$, then $S$ is connected and $d_{\mathrm{sECR}}(T, T') = d_{\mathrm{ECR}}(T, T')$. Hence, the statement holds for $\widetilde{T} = T'$. Otherwise, let $\widetilde{S}$ be an inclusion-maximal subset of $S$, such that $\widetilde{S}$ forms a subtree of $T$. Since $\widetilde{S}$ is inclusion-maximal, we can obtain $T'$ from $T$ in two steps: First, we can obtain an intermediate $X$-tree $\widetilde{T}$ from $T$ by an sECR operation with contraction set $\widetilde{S}$. Second, we can obtain $T'$ from $\widetilde{T}$ by an ECR operation with contraction set $S \setminus \widetilde{S}$.

Before we show that $d_{\mathrm{ECR}}(T, T') \geq d_{\mathrm{SPR}}(T, T')$, we first show the statement for $d_{\mathrm{sECR}}$, that is, we show that $d_{\mathrm{sECR}}(T, T') \geq d_{\mathrm{SPR}}(T, T')$.

**Lemma 6.6.** *Let $T$ and $T'$ be binary $X$-trees. Then, $d_{\mathrm{sECR}}(T, T') \geq d_{\mathrm{SPR}}(T, T')$.*

*Proof.* Let $k = d_{\mathrm{sECR}}(T, T')$. Hence, there is a set $S$ of $k$ internal edges in $T$ such that $T'$ can be obtained by an sECR operation with contraction set $S$. Let $V'$ be the vertices of $T$ incident with some edge of $S$ and let $V^*$ be the neighbors of $V'$ in $T$ that are not incident with any edge of $S$. Recall that by definition of sECR operations, the edges of $S$ induce a subtree of $T$. Hence, $T(V^*)$ is a binary $V^*$-tree having the set $S$ as internal edges. For each vertex $v$ of $V^*$, let $T_v$ denote the pendant subtree of $T$ with pseudo-root $v$ obtained by removing the edge between $v$ and the unique neighbor of $v$ in $V'$. Since $T'$ can be obtained by an sECR operation with contraction set $S$, $T'$ contains a subtree $T'_v$ isomorphic to $T_v$ for each vertex $v$ of $V^*$. Hence, $d_{\mathrm{SPR}}(T, T') = d_{\mathrm{SPR}}(T_S, T'_S)$, where $T_S$ is obtained from $T$ by replacing $T_v$ by the auxiliary taxa $v$ for each vertex $v$ of $V^*$ and where $T'_S$ is obtained from $T'$ by replacing $T'_v$ by the auxiliary taxa $v$ for each vertex $v$ of $V^*$ [3]. Note that $T_S = T(V^*)$.

Hence, it remains to show that $d_{\mathrm{SPR}}(T_S, T'_S) \leq k$. Since $T$ is binary and the edges of $S$ induce a subtree of $T$, $|V^*| = |S| + 3$. Moreover, since for each set of taxa $X'$ and each two binary $X'$-trees $\widetilde{T}$ and $\widehat{T}$, $d_{\mathrm{SPR}}(\widetilde{T}, \widehat{T}) \leq |X'| - 3$ [3], we conclude $d_{\mathrm{SPR}}(T_S, T'_S) \leq |V^*| - 3 = |S| = k$. Consequently, $d_{\mathrm{SPR}}(T, T') \leq k = d_{\mathrm{sECR}}(T, T')$. □

Due to Observation 6.5 and Lemma 6.6, we are now ready to prove that the SPR-distance between two binary $X$-trees is never larger than the ECR-distance between these binary $X$-trees.

**Lemma 6.7.** *Let $T$ and $T'$ be binary $X$-trees. Then, $d_{\mathrm{ECR}}(T, T') \geq d_{\mathrm{SPR}}(T, T')$.*

*Proof.* We show this statement by induction over $d_{\mathrm{ECR}}(T, T')$. For the base case, consider $d_{\mathrm{ECR}}(T, T') \leq 1$. Since $d_{\mathrm{ECR}}(T, T') = 0$ implies that $T$ is isomorphic to $T'$, $d_{\mathrm{SPR}}(T, T') = 0$ if $d_{\mathrm{ECR}}(T, T') = 0$. Moreover, since the 1-ECR operations are exactly the NNI operations and each NNI operation is an SPR operation, $d_{\mathrm{SPR}}(T, T') = 1$ if $d_{\mathrm{ECR}}(T, T') = 1$.

For the inductive step, suppose that $d_{\mathrm{ECR}}(T, T') > 1$ and that the statement holds for each pair of binary $X$-trees $\widehat{T}$ and $\widetilde{T}$ with $d_{\mathrm{ECR}}(\widehat{T}, \widetilde{T}) < d_{\mathrm{ECR}}(T, T')$. Due to Observation 6.5, there is a binary $X$-tree $\widetilde{T}$ with $d_{\mathrm{sECR}}(\widetilde{T}, T') > 0$ such that $d_{\mathrm{ECR}}(T, T') = d_{\mathrm{ECR}}(T, \widetilde{T}) + d_{\mathrm{sECR}}(\widetilde{T}, T')$. Moreover, Lemma 6.6 now implies that $d_{\mathrm{sECR}}(\widetilde{T}, T') \geq d_{\mathrm{SPR}}(\widetilde{T}, T')$. Hence, if $\widetilde{T}$ is isomorphic to $T$, then $d_{\mathrm{ECR}}(T, T') = d_{\mathrm{sECR}}(\widetilde{T}, T') \geq d_{\mathrm{SPR}}(\widetilde{T}, T')$ and the statement holds. Otherwise, that is, if $\widetilde{T}$ is non-isomorphic to $T$, then $d_{\mathrm{ECR}}(T, \widetilde{T}) > 0$. Since $d_{\mathrm{sECR}}(\widetilde{T}, T') > 0$, $d_{\mathrm{ECR}}(T, \widetilde{T}) < d_{\mathrm{ECR}}(T, T')$. Hence, by the induction hypothesis, $d_{\mathrm{SPR}}(T, \widetilde{T}) \leq d_{\mathrm{ECR}}(T, \widetilde{T})$. Since the distance measure $d_{\mathrm{SPR}}$ is a metric, we conclude

$$\begin{aligned} d_{\mathrm{SPR}}(T, T') &\leq d_{\mathrm{SPR}}(T, \widetilde{T}) + d_{\mathrm{SPR}}(\widetilde{T}, T') \\ &\leq d_{\mathrm{ECR}}(T, \widetilde{T}) + d_{\mathrm{sECR}}(\widetilde{T}, T') = d_{\mathrm{ECR}}(T, T'). \end{aligned}$$

Hence, the statement holds. □

Consequently, $d_{\mathrm{ECR}}$ is a metric, upper bounded by $d_{\mathrm{NNI}}$ and lower bounded by $d_{\mathrm{TBR}}$. Moreover, due to the relation between $d_{\mathrm{ECR}}$ and $d_{\mathrm{RF}}$, we obtain the following relation between the considered distance measures.

**Theorem 6.8.** *Let $T$ and $T'$ be binary $X$-trees. Then, $d_{\mathrm{NNI}}(T, T') \geq d_{\mathrm{ECR}}(T, T') = \frac{1}{2} \cdot d_{\mathrm{RF}}(T, T') \geq d_{\mathrm{SPR}}(T, T') \geq d_{\mathrm{TBR}}(T, T')$.*

## 6.3 Hardness of Local Search for the Maximum Parsimony Problem

In this section, we establish our running time lower bounds for $d$-LS Maximum Parsimony.

**Theorem 6.9.** *For each distance measure $d \in \{d_{\mathrm{NNI}}, d_{\mathrm{ECR}}, d_{\mathrm{RF}}, d_{\mathrm{SPR}}, d_{\mathrm{TBR}}\}$ and even if each character is binary, $d$-LS Maximum Parsimony*

- *is* NP*-complete,*
- W[1]*-hard when parameterized by $k$, and*

- *cannot be solved in $f(k) \cdot |I|^{o(k)}$ time for any computable function $f$, unless the ETH fails.*

We reduce from Clique. Recall that Clique is NP-hard [97], W[1]-hard when parameterized by $k$ [44], and cannot be solved in $f(k) \cdot |I|^{o(k)}$ time for any computable function $f$, unless the ETH fails [33, 35].

Let $I = (G = (V, E), k)$ be an instance of Clique and let $d$ be any distance measure from $\{d_{\text{NNI}}, d_{\text{ECR}}, d_{\text{SPR}}, d_{\text{TBR}}\}$. We describe how to construct an equivalent instance $I' = (X, T = (V', E'), S, k')$ of $d$-LS Maximum Parsimony in polynomial time where $k' := k$ if $d \in \{d_{\text{SPR}}, d_{\text{TBR}}\}$ and $k' := 2k$ if $d \in \{d_{\text{NNI}}, d_{\text{ECR}}\}$.

**Definition of $X$ and $T$.** We start with an empty taxa set $X$ and add for each vertex $v \in V$, a set $X_v$ consisting of the eight taxa

$$\text{in}_v^0, \text{in}_v^1, \overline{\text{in}}_v^0, \overline{\text{in}}_v^1, \overline{\text{out}}_v^0, \overline{\text{out}}_v^1, \text{out}_v^0, \text{and } \text{out}_v^1$$

to $X$. Additionally, we add a taxon $x^*$ to $X$. This completes the definition of $X$.

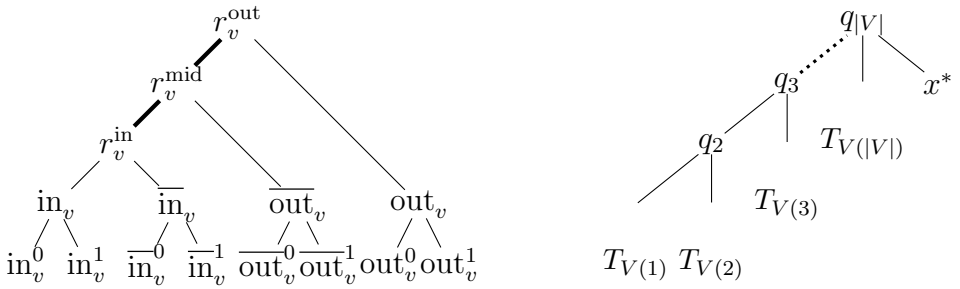
Next, we define the binary $X$-tree $T = (V', E')$. Since $X$ contains $8 \cdot |V| + 1$ taxa and each internal vertex of $T$ has three neighbors, $T'$ has $16 \cdot |V|$ vertices and $2 \cdot |X| - 3 = 16 \cdot |V| - 1$ edges. By definition, $V'$ is a superset of $X$. Additionally, for each vertex $v \in V$, the set $V'$ contains the seven vertices

$$\text{in}_v, \overline{\text{in}}_v, \overline{\text{out}}_v, \text{out}_v, r_v^{\text{in}}, r_v^{\text{mid}}, \text{and } r_v^{\text{out}}.$$

The subtree $T_v := T(X_v)$ is depicted in Figure 6.2a.

Moreover, $V'$ contains $|V| - 1$ additional vertices $q_i$ with $i \in [2, |V|]$. Fix some arbitrary ordering of the vertices of $V$ and let $V(i)$ denote the $i$th vertex of $V$ according to that ordering. The vertex $q_2$ is adjacent to $r_{V(1)}^{\text{out}}, r_{V(2)}^{\text{out}}$, and $q_3$. For each $i \in [3, |V| - 1]$, the vertex $q_i$ is adjacent to $q_{i-1}, q_{i+1}$, and $r_{V(i)}^{\text{out}}$. Finally, $q_{|V|}$ is adjacent to $q_{|V|-1}$, $r_{V(|V|)}^{\text{out}}$, and $x^*$. See Figure 6.2b for an illustration. This completes the definition of $T$.

**Intuition.** The idea of the reduction is as follows: Some of the characters that we define in the following will ensure that each binary $X$-tree $T'$ that improves over $T$ contains a pendant subtree $T'(X_v)$ for each vertex $v \in V$. Further characters will ensure that there are only two non-isomorphic options for $T'(X_v)$ which are depicted in Figure 6.2a and Figure 6.3. Intuitively, these two choices then function as a selection gadget for selecting vertex $v$ as a vertex of the sought clique $K$. The budget $k'$ bounds how many such vertices can be selected. Finally, further characters will ensure that $T'$ improves over $T$ only if $E(K)$ contains at least $\binom{k}{2}$ edges.




**(a)** For a vertex $v \in V$, the pendant $X_v$-tree $T_v$. The bold edges are the only edges of $T_v$ that are not in $R$. **(b)** The subtree of $T$ connecting the pendant trees $T_v$ for each vertex $v \in V$.

**Figure 6.2:** The construction of the $X$-tree $T$.

**Definition of the characters of $S$.** Next, we define the characters of $S$ which are all binary characters whose character states are 0 and 1. We obtain $S$ by concatenating two sequences of characters, $S_G$ and $S_R$, which we describe in the following.

First, we describe the characters of $S_G$. An overview of the characters is given in Table 6.1. We initialize $S_G$ as the empty sequence.

For each edge $e \in E$, we add a character $c_e$ to $S_G$. Let $e$ be an edge of $E$. We set $c_e(x^*) := 1$. Let $v$ be a vertex of $V$. If $v$ is an endpoint of $e$, we set $c_e(x) := 1$ for each taxon $x \in \{\text{in}_v^0, \text{in}_v^1, \overline{\text{in}}_v^0, \overline{\text{in}}_v^1\}$ and we set $c_e(x) := 0$ for each taxon $x \in \{\overline{\text{out}}_v^0, \overline{\text{out}}_v^1, \text{out}_v^0, \text{out}_v^1\}$. Otherwise, if $v$ is not an endpoint of $e$, we set $c_e(x) := 1$ for each taxon $x \in \{\text{in}_v^1, \overline{\text{in}}_v^1, \overline{\text{out}}_v^1, \text{out}_v^1\}$ and we set $c_e(x) := 0$ for each taxon $x \in \{\text{in}_v^0, \overline{\text{in}}_v^0, \overline{\text{out}}_v^0, \text{out}_v^0\}$. Let $S_E$ denote the sequence of characters $c_e$ for each edge $e \in E$.

Next, we define a character $c_{\text{mal}}$. We set $c_{\text{mal}}(x^*) := 1$. For each vertex $v \in V$, we set $c_{\text{mal}}(\text{out}_v^0) = c_{\text{mal}}(\text{out}_v^1) := 1$ and we set $c_{\text{mal}}(x) := 0$ for each taxon $x \in X_v \setminus \{\text{out}_v^0, \text{out}_v^1\}$. We add a sequence $S_{\text{mal}}$ of $\binom{k}{2} - 1$ copies of $c_{\text{mal}}$ to $S_G$. Intuitively, in a binary $X$-tree $T'$, if both endpoints of an edge $e \in E$ are in the selected set $K$, then the parsimony score of $c_e$ in $T'$ is exactly the parsimony score of $c_e$ in $T$ minus one. Moreover, if $T'$ is non-isomorphic to $T$, then the parsimony score of $S_{\text{mal}}$ in $T'$ is exactly the parsimony score of $S_{\text{mal}}$ in $T$ plus $|S_{\text{mal}}|$. Hence, the characters of $S_{\text{mal}}$ act as a hurdle to ensure that $E(K)$ contains at least $|S_{\text{mal}}| + 1 = \binom{k}{2}$ edges.

Finally, for each vertex $v \in V$, we define four characters $c_{v,\text{in}}$, $c_{v,\text{out}}$, $c_{v,\text{ri}}$, and $c_{v,\text{ro}}$. For each taxon $x$ of $X \setminus X_v$, we set $c_{v,\text{in}}(x) := c_{v,\text{out}}(x) := c_{v,\text{ri}}(x) := c_{v,\text{ro}}(x) := 1$. Now, let $x$ be a taxon of $X_v$.

- If $x$ is in $\{\text{in}_v^0, \text{in}_v^1\}$, then we set $c_{v,\text{in}}(x) := 1$, $c_{v,\text{out}}(x) := 0$, $c_{v,\text{ri}}(x) := 1$, and $c_{v,\text{ro}}(x) := 0$.

- If $x$ is in $\{\overline{\text{in}}_v^0, \overline{\text{in}}_v^1\}$, then we set $c_{v,\text{in}}(x) := 1$, $c_{v,\text{out}}(x) := 0$, $c_{v,\text{ri}}(x) := 0$, and $c_{v,\text{ro}}(x) := 0$.
- If $x$ is in $\{\overline{\text{out}}_v^0, \overline{\text{out}}_v^1\}$, then we set $c_{v,\text{in}}(x) := 0$, $c_{v,\text{out}}(x) := 1$, $c_{v,\text{ri}}(x) := 0$, and $c_{v,\text{ro}}(x) := 0$.
- If $x$ is in $\{\text{out}_v^0, \text{out}_v^1\}$, then we set $c_{v,\text{in}}(x) := 0$, $c_{v,\text{out}}(x) := 1$, $c_{v,\text{ri}}(x) := 0$, and $c_{v,\text{ro}}(x) := 1$.

Let $\beta := 2{\cdot}|X|{\cdot}(|E|+\binom{k}{2})$. Note that $\beta$ is larger than $\text{score}_{S_E}(T')+\text{score}_{S_{\text{mal}}}(T')$ of any binary $X$-tree $T'$, since such a tree $T'$ contains less than $2{\cdot}|X|$ edges and $|S_E|+|S_{\text{mal}}| = |E| + \binom{k}{2} - 1$. For each vertex $v \in V$, we extend $S_G$ by

- a sequence $S_{v,\text{in}}$ of $\beta$ copies of $c_{v,\text{in}}$,
- a sequence $S_{v,\text{out}}$ of $\beta$ copies of $c_{v,\text{out}}$,
- a sequence $S_{v,\text{ri}}$ of $2\beta$ copies of $c_{v,\text{ri}}$, and
- a sequence $S_{v,\text{ro}}$ of $2\beta$ copies of $c_{v,\text{ro}}$.

Let $S_v$ denote the combined sequences of $S_{v,\text{in}}$, $S_{v,\text{out}}$, $S_{v,\text{ri}}$, and $S_{v,\text{ro}}$. Intuitively, for each binary $X$-tree $T'$ that improves over $T$ and contains $T'(X_v)$ as a pendant subtree, the characters of $S_v$ ensure that $T'(X_v)$ is isomorphic to either the pendant tree depicted in Figure 6.2a or the pendant tree depicted in Figure 6.3. These two choices then function as a selection gadget for the vertices of the sought clique in $G$. This completes the construction of $S_G$. Note that $S_G$ contains exactly $|E|+\binom{k}{2}-1+6\beta{\cdot}|V|$ characters.

Next, we describe the sequence of characters $S_R$. Let $\alpha := 2 \cdot |X| \cdot |S_G|$. Note that $\alpha$ is larger than $\text{score}_{S_G}(\widetilde{T})$ of any binary $X$-tree $\widetilde{T}$, since such a tree $\widetilde{T}$ contains less than $2 \cdot |X|$ edges. Let $R := E' \setminus \{\{r_v^{\text{in}}, r_v^{\text{mid}}\}, \{r_v^{\text{mid}}, r_v^{\text{out}}\} \mid v \in V\}$. For each edge $e$ of $R$, we define a character $c_R^e$. Let $A|B$ be the split of $T$ induced by $e$. For each taxon $x \in A$, we set $c_R^e(x) := 0$ and for each taxon $x \in B$, we set $c_R^e(x) := 1$. We add as sequence $S_R^e$ of $\alpha$ copies of $c_R^e$ to $S_R$. Intuitively, the characters of $S_R$ ensure that each binary $X$-tree $T'$ that improves over $T$, shares the split that is induced by $e$ in $T$ for each edge $e$ of $R$. This implies that $T'(X_v)$ is a pendant subtree of $T'$ for each vertex $v \in V$.

Finally, $S$ is obtained by concatenating $S_G$ and $S_R$.

**Properties of binary $X$-trees.** Before we show the correctness of the reduction, we first make some observations about binary $X$-trees with the characters of the construction.

**Table 6.1:** An overview of the characters of $S_G$.

| | $\text{in}_v^0$ | $\text{in}_v^1$ | $\overline{\text{in}}_v^0$ | $\overline{\text{in}}_v^1$ | $\overline{\text{out}}_v^0$ | $\overline{\text{out}}_v^1$ | $\text{out}_v^0$ | $\text{out}_v^1$ | $x^*$ |
|---|---|---|---|---|---|---|---|---|---|
| $c_e \in S_E, v \in e$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| $c_e \in S_E, v \notin e$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| $c_{\text{mal}}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| $c_{v,\text{in}}$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| $c_{v,\text{out}}$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| $c_{v,\text{ri}}$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $c_{v,\text{ro}}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| $c \in S_w, w \neq v$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Note that for each binary $X$-tree $T'$ and each edge $e$ of $R$, $\text{score}_{c_R^e}(T') \geq 1$.

**Definition 6.10.** Let $T'$ be a binary $X$-tree. We say that $T'$ is *split-consistent for $T$ and $R$* if for each edge $e$ of $R$, the split of $T$ induced by $e$ is also a split of $T'$.

In preparation for the next observation, note that if a binary $X$-tree $T'$ is not split-consistent for $T$ and $R$, then there is some edge $e$ of $R$ such that $\text{score}_{c_R^e}(T') \geq 2$ and thus $\text{score}_{S_R^e}(T') \geq 2 \cdot \alpha$. Hence, $\text{score}_S(T') \geq \text{score}_{S_R}(T') \geq \alpha \cdot (|R| + 1)$. Since $\alpha > \text{score}_{S_G}(T)$, this implies $\text{score}_S(T') > \text{score}_S(T)$. Hence, we conclude the following.

**Observation 6.11.** *Let $T'$ be a binary $X$-tree. a) If $\text{score}_S(T') \leq \text{score}_S(T)$, then $T'$ is split-consistent for $T$ and $R$. b) If $T'$ is split-consistent for $T$ and $R$, then $\text{score}_{S_R}(T') = \alpha \cdot |R|$.*

To determine whether $I'$ is a yes-instance of $d$-LS Maximum Parsimony, we analyze the structure of binary $X$-trees $T'$ with $\text{score}_S(T') \leq \text{score}_S(T)$. Due to Observation 6.11, we only need to consider binary $X$-trees that are split-consistent for $T$ and $R$ in the following.

Let $v$ be a vertex of $V$ and let $T'$ be a binary $X$-tree which is split-consistent for $T$ and $R$. Since there is an edge $e_v$ in $T$ such that $e_v$ induces the split $X_v|(X \setminus X_v)$ in $T$ and $e_v$ is contained in $R$, $X_v|(X \setminus X_v)$ is a split in $T'$. Hence, $T'(X_v)$ is a pendant tree. Moreover, since all edges incident with $\text{in}_v$ are in $R$, we can assume that $\text{in}_v$ is the common neighbor of $\text{in}_v^0$ and $\text{in}_v^1$ in $T'$. Similarly, we may assume that $\overline{\text{in}}_v$ is the common neighbor of $\overline{\text{in}}_v^0$ and $\overline{\text{in}}_v^1$ in $T'$, $\overline{\text{out}}_v$ is the common neighbor of $\overline{\text{out}}_v^0$ and $\overline{\text{out}}_v^1$ in $T'$, and $\text{out}_v$ is the common neighbor of $\text{out}_v^0$ and $\text{out}_v^1$ in $T'$.

**Definition 6.12.** Let $T'$ be a binary $X$-tree which is split-consistent for $T$ and $R$, let $v$ be a vertex of $V$, and let $r$ be the pseudo-root of the pendant tree $T'(X_v)$.

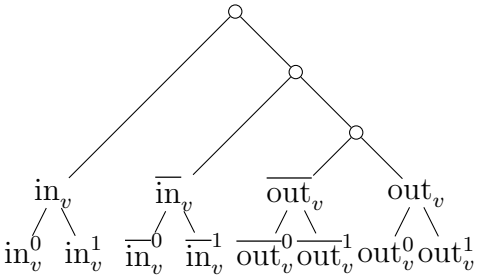



**Figure 6.3:** An in-rooting of $T_v$.

We say that $T'(X_v)$ is an *in-rooting* of $T_v$ if $\text{in}_v$ is adjacent to $r$, $\overline{\text{in}}_v$ has distance 2 to $r$, and both $\overline{\text{out}}_v$ and $\text{out}_v$ have distance 3 to $r$. Similarly, we say that $T'(X_v)$ is an *out-rooting* of $T_v$ if $\text{out}_v$ is adjacent to $r$, $\overline{\text{out}}_v$ has distance 2 to $r$, and both $\overline{\text{in}}_v$ and $\text{in}_v$ have distance 3 to $r$.

Figure 6.2a shows an out-rooting of $T_v$ and Figure 6.3 shows an in-rooting of $T_v$.

Note that for each vertex $v$ of $V$, there is a unique in-rooting of $T_v$ with respect to isomorphism. Similarly, there is a unique out-rooting of $T_v$ with respect to isomorphism. Note that for each vertex $v \in V$, $T_v$ is an out-rooting of $T_v$. We call a binary $X$-tree $T'$ *well-rooted* if $T'$ is split-consistent for $T$ and $R$ and if for each vertex $v \in V$, $T'(X_v)$ is either an in-rooting or an out-rooting of $T_v$. Note that $T$ is well-rooted.

**Lemma 6.13.** *Let $T'$ be a binary $X$-tree which is split-consistent for $T$ and $R$ and let $v$ be a vertex of $V$. If $T'(X_v)$ is an in-rooting or an out-rooting of $T_v$, then* $\text{score}_{S_v}(T') = 9\beta$. *Otherwise,* $\text{score}_{S_v}(T') \geq 10\beta$.

*Proof.* Let $T'_v := T'(X_v)$. Recall that $T'_v$ is a pendant tree in $T'$. Let $r'$ be the pseudo-root of $T'_v$ and let $q$ be the unique neighbor of $r'$ outside of $T'_v$ in $T'$.

In the following, consider the sequence $\widehat{S}$ of characters containing one copy of both $c_{v,\text{in}}$ and $c_{v,\text{out}}$, and two copies of both $c_{v,\text{ri}}$ and $c_{v,\text{ro}}$. Note that $S_v$ consists of $\beta$ copies of $\widehat{S}$. Hence, it remains to show that, if $T'_v$ is either an in-rooting of $T_v$ or an out-rooting of $T_v$, then $\text{score}_{\widehat{S}}(T') = 9$ and $\text{score}_{\widehat{S}}(T') \geq 10$, otherwise.

Recall Observation 6.1: If all taxa $X'$ of a pendant subtree $T(X')$ receive the same image according to a given character $c$, then for each optimal extension $c^*$ of $c$, each vertex of $T(X)$ receives the same label under $c^*$. Note that $\text{in}_v$ is the common neighbor of both $\text{in}_v^0$ and $\text{in}_v^1$ in $T'$. Moreover, since $c(\text{in}_v^0) = c(\text{in}_v^1)$ for each character $c$

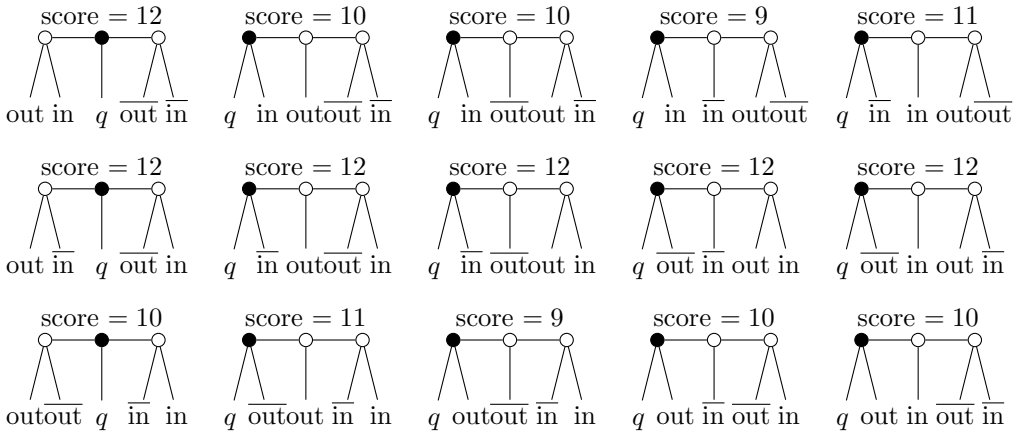



**Figure 6.4:** The 15 potential pairwise non-isomorphic subtrees and their corresponding scores described in the proof of Lemma 6.13. In each tree, the pseudo-root of $T'(X_v)$ is highlighted by the black vertex.

of $\widehat{S}$, Observation 6.1 implies that $c^*(\text{in}_v) = c(\text{in}_v^0) = c(\text{in}_v^1)$. Similarly, for each character $c$ of $\widehat{S}$, $c^*(\overline{\text{in}}_v) = c(\overline{\text{in}}_v^0) = c(\overline{\text{in}}_v^1)$, $c^*(\overline{\text{out}}_v) = c(\overline{\text{out}}_v^0) = c(\overline{\text{out}}_v^1)$, and $c^*(\text{out}_v) = c(\text{out}_v^0) = c(\text{out}_v^1)$. Finally, since the edge $\{r', q\}$ induces the split $X_v|(X \setminus X_v)$ in $T'$ and for each character $c$ of $\widehat{S}$ and each taxon $x \in X \setminus X_v$, $c(x) = 1$, Observation 6.1 implies $c^*(q) = 1$. As a consequence, for each character $c$ of $\widehat{S}$, each mutation edge of $c^*$ is an edge of the subtree $\widehat{T} := T'(\{\text{in}_v, \overline{\text{in}}_v, \overline{\text{out}}_v, \text{out}_v, q\})$. Since both $\widehat{T}$ and $\widehat{S}$ have constant size, we omit the proof of the scores of the 15 different non-isomorphic potential trees $\widehat{T}$. Instead, each such tree together with the corresponding score is depicted in Figure 6.4. □

Next, we describe for a given well-rooted binary $X$-tree $T'$ the maximum parsimony scores of $T'$ with respect to the characters of $S_E$ and $S_{\text{mal}}$. The idea is that in a well-rooted binary $X$-tree $T'$, for each edge $e = \{u, v\} \in E$ where $T'(X_u)$ is an in-rooting of $T_u$ and where $T'(X_v)$ is an in-rooting of $T_v$, the parsimony score of the character $c_e$ in $T'$ is exactly the parsimony score of the character $c_e$ in $T$ minus one. Moreover, if for at least one vertex $v \in V$, $T'(X_v)$ is an in-rooting of $T_v$, then the parsimony score of the characters of $S_{\text{mal}}$ in $T'$ is exactly the parsimony score of the characters of $S_{\text{mal}}$ in $T$ plus $\binom{k}{2} - 1$.

**Lemma 6.14.** *Let $T'$ be a well-rooted binary $X$-tree.*

*a) Let $e = \{u, v\}$ be an edge of $E$. If $T'(X_u)$ is an in-rooting of $T_u$ and $T'(X_v)$ is*

*an in-rooting of $T_v$, then* $\text{score}_{c_e}(T') = 4(|V|-2)+2$. *Otherwise,* $\text{score}_{c_e}(T') = 4(|V|-2)+3$.

*b) If there is at least one vertex $w \in V$ such that $T'(X_w)$ is an in-rooting of $T_w$, then* $\text{score}_{c_{\text{mal}}}(T') = |V|+1$. *Otherwise, that is, if $T'$ is isomorphic to $T$, then* $\text{score}_{c_{\text{mal}}}(T') = |V|$.

*Proof.* For each vertex $w$ of $V$, let $T'_w := T'(X_w)$. Let $V_{\text{in}}$ be those vertices $w$ of $V$, where $T'_w$ is an in-rooting of $T_w$ and let $V_{\text{out}} = V \setminus V_{\text{in}}$ be those vertices $w$ of $V$, where $T'_w$ is an out-rooting of $T_w$. For each vertex $w \in V_{\text{in}}$, let $r_w^{\text{in}}$ be the name of the pseudo-root of $T'_w$, let $r_w^{\text{mid}}$ and $\text{in}_w$ be the neighbors of $r_w^{\text{in}}$, and let $r_w^{\text{out}}$ and $\overline{\text{in}}_w$ be the neighbors of $r_w^{\text{mid}}$. Analogously, for each vertex $w \in V_{\text{out}}$, let $r_w^{\text{out}}$ be the name of the pseudo-root of $T'_w$, let $r_w^{\text{mid}}$ and $\text{out}_w$ be the neighbors of $r_w^{\text{out}}$, and let $r_w^{\text{in}}$ and $\overline{\text{out}}_w$ be the neighbors of $r_w^{\text{mid}}$. Recall that since $T'$ is well-rooted, for each vertex $w \in V$, $\text{in}_w$ is adjacent to both $\text{in}_w^0$ and $\text{in}_w^1$, $\overline{\text{in}}_w$ is adjacent to both $\overline{\text{in}}_w^0$ and $\overline{\text{in}}_w^1$, $\overline{\text{out}}_w$ is adjacent to both $\overline{\text{out}}_w^0$ and $\overline{\text{out}}_w^1$, and $\text{out}_w$ is adjacent to both $\text{out}_w^0$ and $\text{out}_w^1$.

First, we show statement a). Let $c_e$ be a character for some edge $e = \{u, v\}$ of $E$. For each vertex $w \in V \setminus \{u, v\}$,

- let $P_{\text{in}_w}$ be the unique path between $\text{in}_w^0$ and $\text{in}_w^1$ in $T'$,
- let $P_{\overline{\text{in}}_w}$ be the unique path between $\overline{\text{in}}_w^0$ and $\overline{\text{in}}_w^1$ in $T'$,
- let $P_{\overline{\text{out}}_w}$ be the unique path between $\overline{\text{out}}_w^0$ and $\overline{\text{out}}_w^1$ in $T'$, and
- let $P_{\text{out}_w}$ be the unique path between $\text{out}_w^0$ and $\text{out}_w^1$ in $T'$.

Note that each of these four paths only contains two edges and that these four paths are pairwise edge-disjoint. Let $\mathcal{P}_w := \{P_{\text{in}_w}, P_{\overline{\text{in}}_w}, P_{\overline{\text{out}}_w}, P_{\text{out}_w}\}$. Let $P'$ be a path in $\mathcal{P}_w$ and let $w^0$ and $w^1$ be the terminals of $P'$. Since by definition $c_e(w^0) \neq c_e(w^1)$, for each extension $c_e^*$ of $c_e$ in $T'$, at least one edge of $P'$ is a mutation edge of $c_e^*$. Note that each path in $\mathcal{P}_w$ is edge-disjoint with each path in $\mathcal{P}_{w'}$ for distinct vertices $w$ and $w'$ of $V \setminus \{u, v\}$. Moreover, let $P_u$ be the path between $\overline{\text{in}}_u^0$ and $\overline{\text{out}}_u^0$ in $T'$ and let $P_v$ be the path between $\overline{\text{in}}_v^0$ and $\overline{\text{out}}_v^0$ in $T'$. Note that $P_u$ and $P_v$ are edge-disjoint and that both are edge-disjoint with each path $P_w \in \mathcal{P}_w$ for each vertex $w \in V \setminus \{u, v\}$. Since $c_e(\overline{\text{in}}_u^0) = 0$ and $c_e(\overline{\text{out}}_u^0) = 1$, for each extension $c_e^*$ of $c_e$ in $T'$, at least one edge of $P_u$ is a mutation edge of $c_e^*$. Similarly, since $c_e(\overline{\text{in}}_v^0) = 0$ and $c_e(\overline{\text{out}}_v^0) = 1$, for each extension $c_e^*$ of $c_e$ in $T'$, at least one edge of $P_v$ is a mutation edge of $c_e^*$. Hence, $\text{score}_{c_e}(T') \geq 4(|V|-2)+2$.

**Case 1:** $T'_u$ is an in-rooting of $T_u$ and $T'_v$ is an in-rooting of $T_v$**.** We define an extension $c^*_e$ of $c_e$ in $T'$, such that $\text{score}_{c^*_e}(T') = 4(|V|-2)+2$. This extension is depicted in Figure 6.5. We set $c^*_e(\text{out}_u) := c^*_e(\overline{\text{out}}_u) := c^*_e(r^{\text{out}}_u) := 0$ and $c^*_e(\text{out}_v) := c^*_e(\overline{\text{out}}_v) := c^*_e(r^{\text{out}}_v) := 0$. For each remaining internal vertex $v'$ of $T'$, we set $c^*_e(v') := 1$. Hence, the edge set

$$\{\{r^{\text{out}}_u, r^{\text{mid}}_u\}, \{r^{\text{out}}_v, r^{\text{mid}}_v\}\} \cup \{\{\text{in}^0_w, \text{in}_w\}, \{\overline{\text{in}}^0_w, \overline{\text{in}}_w\}, \{\overline{\text{out}}^0_w, \overline{\text{out}}_w\}, \{\text{out}^0_w, \text{out}_w\} \mid w \in V \setminus \{u,v\}\}$$

contains the mutation edges of $c^*_e$ in $T'$. Consequently, $\text{score}_{c^*_e}(T') = 4(|V|-2)+2$ which implies $\text{score}_{c_e}(T') = 4(|V|-2)+2$.

**Case 2:** $T'_u$ is an out-rooting of $T_u$ or $T'_v$ is an out-rooting of $T_v$**.** Assume without loss of generality that $T'_v$ is an out-rooting of $T_v$. Let $P^*_x$ be the unique path between $\text{out}^0_v$ and $x^*$ in $T'$. Since $c_e(\text{out}^0_v) = 0$ and $c_e(x^*) = 1$, for each extension $c^*_e$ of $c_e$ in $T'$, at least one edge of $P^*_x$ is a mutation edge of $c^*_e$. Note that $P^*_x$ is edge-disjoint with $P_u$ and edge-disjoint with each path $P_w \in \mathcal{P}_w$ for each vertex $w \in V \setminus \{u,v\}$. Moreover, since $T'_v$ is an out-rooting of $T_v$, $P^*_x$ is also edge-disjoint with $P_v$. Hence, $\text{score}_{c_e}(T') \geq 4(|V|-2)+3$. We define an extension $c^*_e$ of $c_e$ in $T'$, such that $\text{score}_{c^*_e}(T') = 4(|V|-2)+3$. To this end, we distinguish whether $T'_u$ is an in-rooting of $T_u$ or an out-rooting of $T_u$. For both cases, the corresponding extension is depicted in Figure 6.5.

**Case 2.1:** $T'_u$ is an in-rooting of $T_u$**.** We set $c^*_e(\text{out}_u) := c^*_e(\overline{\text{out}}_u) := c^*_e(r^{\text{out}}_u) := 0$ and $c^*_e(\text{out}_v) := c^*_e(\overline{\text{out}}_v) := 0$. For each remaining internal vertex $v'$ of $T'$, we set $c^*_e(v') := 1$. Hence, the edge set

$$\{\{r^{\text{out}}_u, r^{\text{mid}}_u\}, \{r^{\text{mid}}_v, \overline{\text{out}}_v\}, \{r^{\text{out}}_v, \text{out}_v\}\} \cup \{\{\text{in}^0_w, \text{in}_w\}, \{\overline{\text{in}}^0_w, \overline{\text{in}}_w\}, \{\overline{\text{out}}^0_w, \overline{\text{out}}_w\}, \{\text{out}^0_w, \text{out}_w\} \mid w \in V \setminus \{u,v\}\}$$

contains the mutation edges of $c^*_e$ in $T'$.

**Case 2.2:** $T'_u$ is an out-rooting of $T_u$**.** We set $c^*_e(\text{in}_u) := c^*_e(\overline{\text{in}}_u) := c^*_e(r^{\text{in}}_u) := 1$ and $c^*_e(\text{in}_v) := c^*_e(\overline{\text{in}}_v) := c^*_e(r^{\text{in}}_v) := 1$. For each remaining internal vertex $v'$ of $T'$, we set $c^*_e(v') := 0$. Let $q_n$ denote the unique neighbor of $x^*$ in $T'$. Hence, the edge set

$$\{\{r^{\text{in}}_u, r^{\text{mid}}_u\}, \{r^{\text{in}}_v, r^{\text{mid}}_v\}, \{x^*, q_n\}\} \cup \{\{\text{in}^1_w, \text{in}_w\}, \{\overline{\text{in}}^1_w, \overline{\text{in}}_w\}, \{\overline{\text{out}}^1_w, \overline{\text{out}}_w\}, \{\text{out}^1_w, \text{out}_w\} \mid w \in V \setminus \{u,v\}\}$$

contains the mutation edges of $c^*_e$ in $T'$.

Hence, in both cases $\text{score}_{c^*_e}(T') = 4(|V|-2)+3$ which implies $\text{score}_{c_e}(T') = 4(|V|-2)+3$.

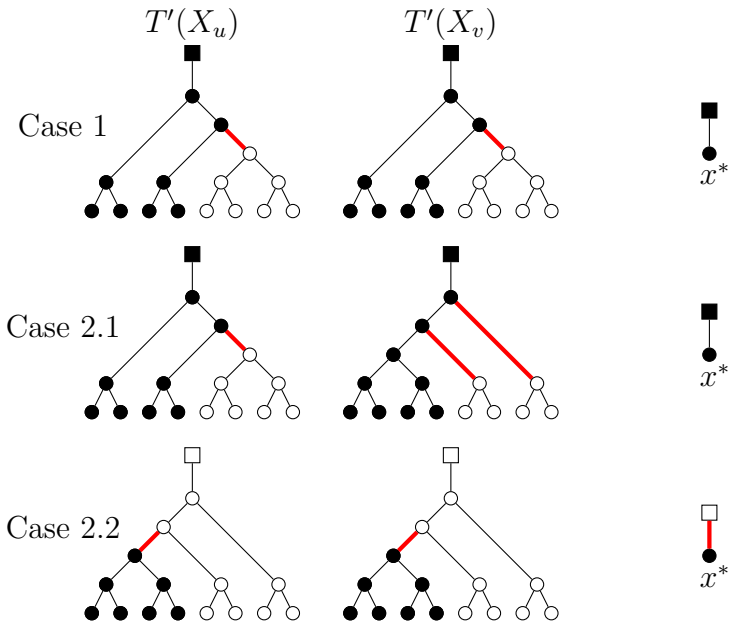



**Figure 6.5:** An optimal extension of $c_{\{u,v\}}$ for each of the three cases in the proof of Lemma 6.14 for any well-rooted binary $X$-tree $T'$. White vertices receive label 0 and black vertices receive label 1 under these extensions. The mutation edges inside the depicted subtrees are highlighted in red. Case 1 shows the extension if $T'(X_u)$ is an in-rooting of $T_u$ and $T'(X_v)$ is an in-rooting of $T_v$. Case 2.1 shows the extension if $T'(X_u)$ is an in-rooting of $T_u$ and $T'(X_v)$ is an out-rooting of $T_v$. Finally, Case 2.2 shows the extension if $T'(X_u)$ is an out-rooting of $T_u$ and $T'(X_v)$ is an out-rooting of $T_v$. For each other vertex $w \in V \setminus \{u, v\}$, the non-depicted subtree $T'(X_w)$ contains exactly four mutation edges, each incident with a leaf. Note that the extension for Case 1 has exactly one mutation edge less than both other extensions.

Next, we show statement b). Consider the character $c_{\text{mal}}$. For each vertex $v \in V$, let $P_v$ be the unique path between $\overline{\text{out}}_v^0$ and $\text{out}_v^0$ in $T'$. Since $c_{\text{mal}}(\overline{\text{out}}_v^0) = 0$ and $c_{\text{mal}}(\text{out}_v^0) = 1$, for each extension $c^*_{\text{mal}}$ of $c_{\text{mal}}$ in $T'$ at least one edge of $P_v$ is a mutation edge of $c^*_{\text{mal}}$. Note that the paths $P_v$ and $P_w$ are edge-disjoint for distinct vertices $v$ and $w$ of $V$. Hence, $\text{score}_{c_{\text{mal}}}(T') \geq |V|$.

**Case 1:** There is some vertex $v \in V$ such that $T'_v$ is an in-rooting of $T_v$**.** Let $P^*_x$ be the unique path between $\text{in}_v^0$ and $x^*$ in $T'$. Since $c_{\text{mal}}(\text{in}_v^0) = 0$ and $c_{\text{mal}}(x^*) = 1$, for each extension $c^*_{\text{mal}}$ of $c_{\text{mal}}$ in $T'$, at least one edge of $P^*_x$ is a mutation edge of $c^*_{\text{mal}}$. Note that $P^*_x$ is edge-disjoint with $P_w$ for each vertex $w \in V$ distinct from $v$. Moreover, since $T'_v$ is an in-rooting of $T_v$, $P^*_x$ is also edge-disjoint

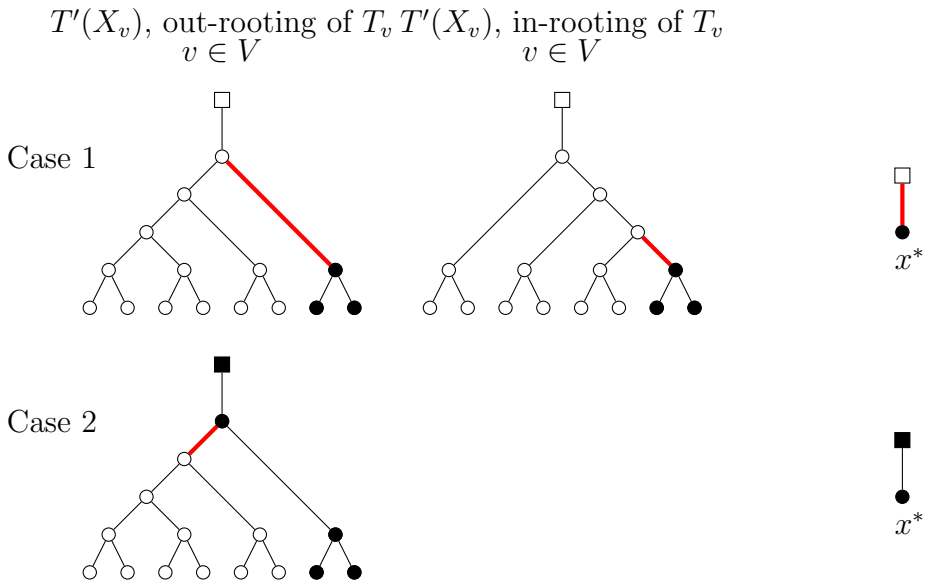



**Figure 6.6:** An optimal extension of $c_{\text{mal}}$ for each of the two cases in the proof of Lemma 6.14 for any well-rooted binary $X$-tree $T'$. White vertices receive label 0 and black vertices receive label 1 under these extensions. The mutation edges inside the depicted subtrees are highlighted in red. Case 1 shows the extension if there is at least one vertex $v \in V$, such that $T'(X_v)$ is an in-rooting of $T_v$. Case 2 shows the extension if for each vertex $v \in V$, $T'(X_v)$ is an out-rooting of $T_v$. Note that the extension for Case 1 has exactly one mutation edge more than the extension of Case 2.

with $P_v$. Hence, $\text{score}_{c_{\text{mal}}}(T') \geq |V| + 1$. We define an extension $c^*_{\text{mal}}$ of $c_{\text{mal}}$ in $T'$, such that $\text{score}_{c^*_{\text{mal}}}(T') = |V| + 1$. This extension is depicted in Figure 6.6. We set $c^*_{\text{mal}}(\text{out}_w) := 1$, for each vertex $w \in V$. For each remaining internal vertex $v'$ of $T'$, we set $c^*_{\text{mal}}(v') := 0$. Let $q_n$ denote the unique neighbor of $x^*$ in $T'$. Hence, the edge set $\{\{q_n, x^*\}\} \cup \{\{\text{out}_v, r_v^{\text{out}}\} \mid v \in V\}$ contains the mutation edges of $c^*_{\text{mal}}$ in $T'$. Consequently, $\text{score}_{c^*_{\text{mal}}}(T') = |V| + 1$ which implies $\text{score}_{c_{\text{mal}}}(T') = |V| + 1$.

**Case 2:** For each vertex $v \in V$, $T'_v$ is an out-rooting of $T_v$**.** Hence, $T'$ is isomorphic to $T$. We define an extension $c^*_{\text{mal}}$ of $c_{\text{mal}}$ in $T'$, such that $\text{score}_{c^*_{\text{mal}}}(T') = |V|$. This extension is depicted in Figure 6.6. We set $c^*_{\text{mal}}(\text{in}_v) := c^*_{\text{mal}}(\overline{\text{in}}_v) := c^*_{\text{mal}}(\text{out}_v) := c^*_{\text{mal}}(r_v^{\text{in}}) := c^*_{\text{mal}}(r_v^{\text{mid}}) := 0$, for each vertex $v \in V$. For each remaining internal vertex $v'$ of $T'$, we set $c^*_{\text{mal}}(v') = 1$. Hence, the edge set $\{\{r_v^{\text{mid}}, r_v^{\text{out}}\} \mid v \in V\}$ contains the mutation edges of $c^*_{\text{mal}}$ in $T'$. Consequently, $\text{score}_{c^*_{\text{mal}}}(T') = |V|$ which implies that $\text{score}_{c_{\text{mal}}}(T') = |V|$.

□

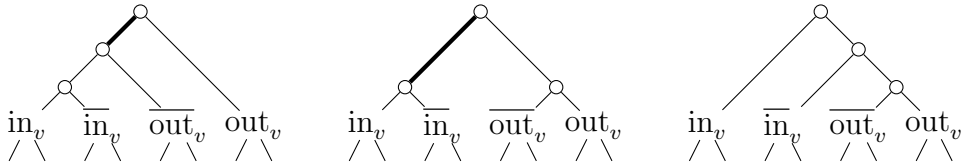



**Figure 6.7:** Two consecutive NNI operation that transform an out-rooting into an in-rooting.

**The score of improving $X$-trees with respect to $S$.** Since $T$ is well-rooted, and for each vertex $v \in V$, $T_v$ is an out-rooting of $T_v$, Observation 6.11, Lemma 6.13, and Lemma 6.14 imply the following.

**Corollary 6.15.** $\text{score}_S(T) = |E| \cdot (4(|V|-2)+3) + (\binom{k}{2}-1) \cdot |V| + |V| \cdot 9\beta + |R| \cdot \alpha$.

Note that by definition, $\beta = 2(8|V|+1) \cdot (|E| + \binom{k}{2}) > |E| \cdot (4(|V|-2)+3) + (\binom{k}{2}-1) \cdot |V|$. Hence, $\text{score}_S(T) < \beta \cdot (9|V|+1) + |R| \cdot \alpha$.

**Corollary 6.16.** *Let $T'$ be a binary $X$-tree with $\text{score}_S(T') < \text{score}_S(T)$. Then, $T'$ is well-rooted.*

*Proof.* Due to Observation 6.11, $T'$ is split-consistent for $T$ and $R$ and $\text{score}_{S_R}(T') = |R| \cdot \alpha$. Assume towards a contradiction that there is a vertex $v \in V$ such that $T'(X_v)$ is neither an in-rooting of $T_v$ nor an out-rooting of $T_v$. Hence, Lemma 6.13 implies $\text{score}_{S_v}(T') \geq 10\beta$ and $\text{score}_{S_w}(T') \geq 9\beta$ for each vertex $w \in V \setminus \{v\}$. Consequently, $\text{score}_S(T') \geq 10\beta + (|V|-1) \cdot 9\beta + |R| \cdot \alpha = \beta \cdot (9|V|+1) + |R| \cdot \alpha > \text{score}_S(T)$, a contradiction. □

**Distances between well-rooted binary $X$-trees.** Next, we describe for each distance measure $d \in \{d_{\text{NNI}}, d_{\text{ECR}}, d_{\text{SPR}}, d_{\text{TBR}}\}$ the distance between $T$ and any other well-rooted binary $X$-tree $T'$.

**Lemma 6.17.** *Let $T'$ be a binary and well-rooted $X$-tree. Moreover, let $K$ be the set of vertices of $V$ such that $T'(X_v)$ is an in-rooting of $T_v$ for each vertex $v \in K$ and $T'(X_w)$ is an out-rooting of $T_w$ for each vertex $w \in V \setminus K$. Then, $d_{\text{NNI}}(T,T') = d_{\text{ECR}}(T,T') = 2 \cdot |K|$ and $d_{\text{SPR}}(T,T') = d_{\text{TBR}}(T,T') = |K|$.*

*Proof.* First, we show that $d_{\text{NNI}}(T,T') = d_{\text{ECR}}(T,T') = 2 \cdot |K|$. To this end, we show that $d_{\text{NNI}}(T,T') \leq 2 \cdot |K|$ and that $d_{\text{ECR}}(T,T') \geq 2 \cdot |K|$. Since $d_{\text{NNI}}(T,T') \geq d_{\text{ECR}}(T,T')$ due to Lemma 6.4, this then implies $d_{\text{NNI}}(T,T') = d_{\text{ECR}}(T,T') = 2 \cdot |K|$.

To show that $d_{\mathrm{NNI}}(T, T') \leq 2 \cdot |K|$, we prove the following: Let $\widetilde{T}$ be a well-rooted binary $X$-tree and let $v$ be a vertex such that $\widetilde{T}(X_v)$ is an out-rooting of $T_v$. Then, $d_{\mathrm{NNI}}(\widetilde{T}, \widehat{T}) \leq 2$, where $\widehat{T}$ is a well-rooted binary $X$-tree with $\widetilde{T}(X \setminus X_v) = \widehat{T}(X \setminus X_v)$ and where $\widehat{T}(X_v)$ is an in-rooting of $T_v$. To show the claim, we describe two consecutive NNI operations transforming $\widetilde{T}$ into $\widehat{T}$. See Figure 6.7 for an illustration of these NNI operations. Let $r_v^{\mathrm{out}}$ be name of the pseudo-root of the pendant tree $\widetilde{T}(X_v)$, let $r_v^{\mathrm{out}}$ be the name of the common neighbor of $r_v^{\mathrm{mid}}$ and $\mathrm{out}_v$ in $\widetilde{T}$, and let $r_v^{\mathrm{mid}}$ be the name of the common neighbor of $r_v^{\mathrm{in}}$ and $\overline{\mathrm{out}}_v$ in $\widetilde{T}$. Moreover, let $q$ be the unique neighbor of $r_v^{\mathrm{out}}$ outside of $\widetilde{T}(X_v)$ in $\widetilde{T}$. We obtain the well-rooted binary $X$-tree $\widehat{T}$ from $\widetilde{T}$ in two steps: First, we remove the edges $\{q, r_v^{\mathrm{out}}\}$ and $\{\overline{\mathrm{out}}_v, r_v^{\mathrm{mid}}\}$ and add the edges $\{\overline{\mathrm{out}}_v, r_v^{\mathrm{out}}\}$ and $\{q, r_v^{\mathrm{mid}}\}$. Second, we remove the edges $\{q, r_v^{\mathrm{mid}}\}$ and $\{\overline{\mathrm{in}}_v, r_v^{\mathrm{in}}\}$ and add the edges $\{\overline{\mathrm{in}}_v, r_v^{\mathrm{mid}}\}$ and $\{q, r_v^{\mathrm{in}}\}$. Since this can be done by two consecutive NNI operations and $\widetilde{T}(X \setminus X_v) = \widehat{T}(X \setminus X_v)$, we conclude $d_{\mathrm{NNI}}(\widetilde{T}, \widehat{T}) \leq 2$. Since $d_{\mathrm{NNI}}$ is a metric one can then show via induction over any arbitrary ordering of the vertices of $K$, that $d_{\mathrm{NNI}}(T, T') \leq 2 \cdot |K|$.

It remains to show that $d_{\mathrm{ECR}}(T, T') \geq 2 \cdot |K|$. Let $\widetilde{E}$ be a subset of the internal edges of $T$, such that $T'$ can be obtained from $T$ by an ECR operation with contraction set $\widetilde{E}$. We show that $|\widetilde{E}| \geq 2 \cdot |K|$. Let $v$ be a vertex of $K$. Recall that $T_v$ is an out-rooting of $T_v$ and that $T'_v$ is an in-rooting of $T_v$. Hence, the edge $\{r_v^{\mathrm{out}}, r_v^{\mathrm{mid}}\}$ induces the split $A|B$ in $T$ with $A := \{\mathrm{in}_v^0, \mathrm{in}_v^1, \overline{\mathrm{in}}_v^0, \overline{\mathrm{in}}_v^1, \overline{\mathrm{out}}_v^0, \overline{\mathrm{out}}_v^1\}$ and $B := X \setminus A$. Since $A|B$ is not a split of $T'$, the edge $\{r_v^{\mathrm{out}}, r_v^{\mathrm{mid}}\}$ is contained in $\widetilde{E}$. Moreover, the edge $\{r_v^{\mathrm{mid}}, r_v^{\mathrm{in}}\}$ induces the split $A|B$ in $T$ with $A := \{\mathrm{in}_v^0, \mathrm{in}_v^1, \overline{\mathrm{in}}_v^0, \overline{\mathrm{in}}_v^1\}$ and $B := X \setminus A$. Since $A|B$ is not a split of $T'$, the edge $\{r_v^{\mathrm{mid}}, r_v^{\mathrm{in}}\}$ is contained in $\widetilde{E}$. Hence, for each vertex $v$ of $V$, $\widetilde{E}$ contains at least two edges of $T(X_v)$. Consequently, $|\widetilde{E}| \geq 2 \cdot |K|$ which implies $d_{\mathrm{ECR}}(T, T') \geq 2 \cdot |K|$.

Second, we show that $d_{\mathrm{SPR}}(T, T') = d_{\mathrm{TBR}}(T, T') = |K|$. Similar to the first part of the proof, we show that $d_{\mathrm{SPR}}(T, T') \leq |K|$ and that $d_{\mathrm{TBR}}(T, T') \geq |K|$. Since $d_{\mathrm{SPR}}(T, T') \geq d_{\mathrm{TBR}}(T, T')$ [3] this then implies $d_{\mathrm{SPR}}(T, T') = d_{\mathrm{TBR}}(T, T') = |K|$.

To show that $d_{\mathrm{SPR}}(T, T') \leq |K|$, we prove the following: Let $\widetilde{T}$ be a well-rooted binary $X$-tree and let $v$ be a vertex such that $\widetilde{T}(X_v)$ is an out-rooting of $T_v$. Then, $d_{\mathrm{SPR}}(\widetilde{T}, \widehat{T}) \leq 1$, where $\widehat{T}$ is a well-rooted binary $X$-tree with $\widetilde{T}(X \setminus X_v) = \widehat{T}(X \setminus X_v)$ and where $\widehat{T}(X_v)$ is an in-rooting of $T_v$.

To show this claim, we describe an SPR operation transforming $\widetilde{T}$ into $\widehat{T}$. See Figure 6.8 for an illustration of this SPR operation. Let $r_v^{\mathrm{out}}$ be the name of the pseudo-root of the pendant tree $\widetilde{T}(X_v)$ and let $q$ be the name of the unique neighbor of $r_v^{\mathrm{out}}$

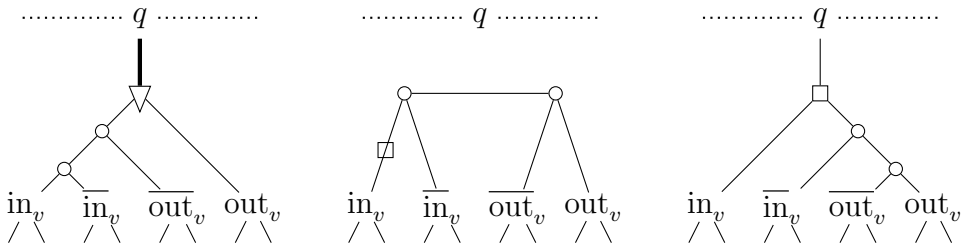



**Figure 6.8:** A transformation of an out-rooting into an in-rooting by a single SPR operation. Firstly, the bold edge is removed and the triangular vertex is suppressed. Secondly, the unique internal edge incident with $\mathrm{in}_v$ is subdivided by the rectangular vertex. Finally, the rectangular vertex is joined with $q$ by a new edge.

outside of $\widetilde{T}(X_v)$ in $\widetilde{T}$. Moreover, let $r_v^{\mathrm{in}}$ be the name of the common neighbor of $\mathrm{in}_v$ and $\overline{\mathrm{in}}_v$ in $\widetilde{T}$. We obtain the well-rooted binary $X$-tree $\widehat{T}$ from $\widetilde{T}$ by: removing the edge $\{r_v^{\mathrm{out}}, q\}$, suppressing the vertex $r_v^{\mathrm{out}}$, subdividing the edge $\{\mathrm{in}_v, r_v^{\mathrm{in}}\}$ by a vertex $q'$, and adding the edge $\{q, q'\}$. Since this can be done by a single SPR operation and $\widetilde{T}(X \setminus X_v) = \widehat{T}(X \setminus X_v)$, we conclude $d_{\mathrm{SPR}}(\widetilde{T}, \widehat{T}) \leq 1$. Since $d_{\mathrm{SPR}}$ is a metric, one can then show via induction over any arbitrary ordering of the vertices of $K$, that $d_{\mathrm{SPR}}(T, T') \leq |K|$.

It remains to show that $d_{\mathrm{TBR}}(T, T') \geq |K|$.

Let $\mathcal{B}$ be an *agreement forest* for $T$ and $T'$, that is, a partition of $X$ such that for each $B \in \mathcal{B}$, $T|_B$ is isomorphic to $T'|_B$ and for each pair of distinct $B \in \mathcal{B}$ and $B' \in \mathcal{B}$, $T(B)$ is vertex disjoint with $T(B')$, and $T'(B)$ is vertex disjoint with $T'(B')$. We show that $|\mathcal{B}| \geq |K| + 1$ which implies $d_{\mathrm{TBR}}(T, T') \geq |K|$ [3]. To this end, we first show that for each vertex $v \in K$, there is some $B \in \mathcal{B}$ with $B \subseteq X_v$.

Assume towards a contradiction, that for some vertex $v \in K$ there is no $B \in \mathcal{B}$ with $B \subseteq X_v$. Let $x_v$ be an arbitrary taxon of $X_v$. Since $\mathcal{B}$ is a partition of $X$, there is some $B_v \in \mathcal{B}$ with $x_v \in B_v$. By assumption, since $B_v$ is not a subset of $X_v$, $B_v \setminus X_v$ is non-empty. In the following, we distinguish whether $B_v$ is a proper superset of $X_v$.

**Case 1:** $B_v$ **is a proper superset of** $X_v$**.** Since $X_v$ is a subset of $B_v$ and $B_v \setminus X_v$ is non-empty, $T(X_v)$ is a subgraph of $T|_{B_v}$ and $T'(X_v)$ is a subgraph of $T'|_{B_v}$. Moreover, since $T'(X_v)$ is an in-rooting of $T_v$, $T(X_v)$ is non-isomorphic to $T'(X_v)$. Thus, $T|_{B_v}$ is non-isomorphic to $T'|_{B_v}$. Hence, $\mathcal{B}$ is not an agreement forest for $T$ and $T'$, a contradiction.

**Case 2:** $B_v$ **is not a proper superset of** $X_v$**.** Hence, there is a taxon $x'_v$ in $X_v \setminus B_v$. Since $\mathcal{B}$ is a partition of $X$, there is some $B'_v \in \mathcal{B}$ with $x'_v \in B'_v$. By assumption, $B'_v$ is not a subset of $X_v$. Hence, $B'_v \setminus X_v$ is non-empty. Since each path in $T$ between any

taxon of $X_v$ and any taxon of $X \setminus X_v$ contains the vertex $r_v^{\text{out}}$, $T(B_v)$ and $T(B'_v)$ are not vertex-disjoint. Hence, $\mathcal{B}$ is not an agreement forest for $T$ and $T'$, a contradiction.

Consequently, for each vertex $v \in K$, there is some $B \in \mathcal{B}$ with $B \subseteq X_v$. Moreover, since $\mathcal{B}$ is a partition of $X$, there is some $B \in \mathcal{B}$ with $x^* \in B$. Hence, $|\mathcal{B}| \geq |K| + 1$ and thus $d_{\text{TBR}}(T, T') \geq |K|$. □

**Correctness.** Finally, we are able to show that $I$ is a yes-instance of Clique if and only if $I'$ is a yes-instance of $d$-LS Maximum Parsimony with appropriate distance bounds.

**Lemma 6.18.** *The following statements are equivalent:*

1. *There is a clique of size $k$ in $G$.*
2. *There is a binary $X$-tree $T'$ with $\text{score}_S(T') < \text{score}_S(T)$ and $d_{\text{SPR}}(T, T') \leq k$.*
3. *There is a binary $X$-tree $T'$ with $\text{score}_S(T') < \text{score}_S(T)$ and $d_{\text{TBR}}(T, T') \leq k$.*
4. *There is a binary $X$-tree $T'$ with $\text{score}_S(T') < \text{score}_S(T)$ and $d_{\text{NNI}}(T, T') \leq 2k$.*
5. *There is a binary $X$-tree $T'$ with $\text{score}_S(T') < \text{score}_S(T)$ and $d_{\text{ECR}}(T, T') \leq 2k$.*

*Proof.* First, we show that Item 1 implies each of the Items 2 to 5. Let $K \subseteq V$ be a clique of size $k$ in $G$. Further, let $T'$ be a well-rooted binary $X$-tree such that for each vertex $v \in K$, $T'(X_v)$ is an in-rooting of $T_v$, and for each vertex $v \in V \setminus K$, $T'(X_v)$ is an out-rooting of $T_v$. Due to Lemma 6.17, $d_{\text{SPR}}(T, T') = d_{\text{TBR}}(T, T') = k$ and $d_{\text{NNI}}(T, T') = d_{\text{ECR}}(T, T') = 2k$. It remains to show that $\text{score}_S(T') < \text{score}_S(T)$. Since $T'$ is well-rooted, due to Observation 6.11, $\text{score}_{S_R}(T') = |R| \cdot \alpha$ and due to Lemma 6.13, for each vertex $v \in V$, $\text{score}_{S_v}(T') = 9\beta$. Moreover, since $K$ is non-empty, we obtain by Lemma 6.14, that $\text{score}_{S_{\text{mal}}}(T') = (\binom{k}{2} - 1) \cdot (|V| + 1)$. Since $K$ is a clique in $G$, $|E(K)| = \binom{k}{2}$. Finally, by Lemma 6.14, for each edge $e$ of $E(K)$, $\text{score}_{c_e}(T') = 4(|V| - 2) + 2$, and for each edge $e$ of $E \setminus E(K)$, $\text{score}_{c_e}(T') = 4(|V| - 2) + 3$. We conclude

$$\begin{aligned}
\text{score}_S(T') &= |E| \cdot (4(|V| - 2) + 3) - \binom{k}{2} + \left(\binom{k}{2} - 1\right) \cdot (|V| + 1) \\
&\quad + |V| \cdot 9\beta + |R| \cdot \alpha \\
&= |E| \cdot (4(|V| - 2) + 3) + \left(\binom{k}{2} - 1\right) \cdot |V| + |V| \cdot 9\beta + |R| \cdot \alpha - 1 \\
&= \text{score}_S(T) - 1,
\end{aligned}$$

due to Corollary 6.15. Hence, $T'$ is a binary $X$-tree with $\text{score}_S(T') < \text{score}_S(T)$, $d_{\text{SPR}}(T,T') = d_{\text{TBR}}(T,T') = k$, and $d_{\text{NNI}}(T,T') = d_{\text{ECR}}(T,T') = 2k$.

Second, we show that each of the Items 2 to 5 implies Item 1. Let $T'$ be a binary $X$-tree with a) $\text{score}_S(T') < \text{score}_S(T)$ and b) $d_{\text{SPR}}(T,T') \leq k$, $d_{\text{TBR}}(T,T') \leq k$, $d_{\text{NNI}}(T,T') \leq 2k$, or $d_{\text{ECR}}(T,T') \leq 2k$. Since $\text{score}_S(T') < \text{score}_S(T)$, $T'$ is well-rooted due to Corollary 6.16, that is, for each vertex $v \in V$, $T'_v := T'(X_v)$ is either an in-rooting of $T_v$ or an out-rooting of $T_v$. Let $K \subseteq V$ be the set of all vertices $v$ of $V$ where $T'_v$ is an in-rooting of $T_v$. We show that $K$ is a clique of size $k$ in $G$. Since $d_{\text{SPR}}(T,T') \leq k$, $d_{\text{TBR}}(T,T') \leq k$, $d_{\text{NNI}}(T,T') \leq 2k$, or $d_{\text{ECR}}(T,T') \leq 2k$, Lemma 6.17 implies that $K$ has size at most $k$. Moreover, since $T'$ is well-rooted, due to Observation 6.11, $\text{score}_{S_R}(T') = |R| \cdot \alpha$ and due to Lemma 6.13, for each vertex $v \in V$, $\text{score}_{S_v}(T') = 9\beta$. Moreover, since $\text{score}_S(T') < \text{score}_S(T)$, $T'$ is not isomorphic to $T$, which implies that $K$ is nonempty. Hence due to Lemma 6.14, $\text{score}_{S_{\text{mal}}}(T') = (\binom{k}{2} - 1) \cdot (|V| + 1)$. Finally, by Lemma 6.14, for each edge $e \in E \setminus E(K)$, $\text{score}_{c_e}(T') = 4(|V| - 2) + 3$, and for each edge $e \in E(K)$, $\text{score}_{c_e}(T') = 4(|V| - 2) + 2$. Consequently, $\text{score}_S(T) - \text{score}_S(T') = |E(K)| - (\binom{k}{2} - 1)$.

Since $\text{score}_S(T') < \text{score}_S(T)$, we have $|E(K)| \geq \binom{k}{2}$, which implies that $K$ is a size-$k$ clique in $G$. □

Since $k' = k$ if $d \in \{d_{\text{SPR}}, d_{\text{TBR}}\}$ and $k' = 2k$ if $d \in \{d_{\text{NNI}}, d_{\text{ECR}}\}$, Lemma 6.18 implies that $I$ is a yes-instance of Clique if and only if $I'$ is a yes-instance of $d$-LS Maximum Parsimony. This completes the proof of Theorem 6.9.

## 6.4 An Adaptation for the Permissive Version

In this section, we describe how to extend the previously described reduction to also work for the permissive version of $d$-LS Maximum Parsimony.

**Theorem 6.19.** *For each distance measure $d \in \{d_{\text{NNI}}, d_{\text{ECR}}, d_{\text{RF}}, d_{\text{SPR}}, d_{\text{TBR}}\}$ and even if each character is binary, the* permissive version *of* $d$-LS Maximum Parsimony

- *is* NP*-complete,*
- *does not admit an* FPT*-algorithm when parameterized by $k$, unless* FPT = W[1]*, and*
- *cannot be solved in $f(k) \cdot |I|^{o(k)}$ time for any computable function $f$, unless the ETH fails.*

**Table 6.2:** An overview of the characters of $S_{\text{mcc}}$ for some vertex $v \in V_i$.

| | $\text{in}_v^0$ | $\text{in}_v^1$ | $\overline{\text{in}}_v^0$ | $\overline{\text{in}}_v^1$ | $\overline{\text{out}}_v^0$ | $\overline{\text{out}}_v^1$ | $\text{out}_v^0$ | $\text{out}_v^1$ | $x_0^*$ | $x_1^*$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_{v,\text{mcc}}$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| $c_{u,\text{mcc}}, u \in V_i \setminus \{v\}$ | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| $c_{w,\text{mcc}}, w \in V \setminus V_i$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

*Proof.* Instead of reducing from Clique, we reduce from Multicolored Clique, which preserves the same required lower bound results as Clique. Here, the difference is that we are additionally given a $k$-partition $(V_1, \ldots, V_k)$ of the vertex set $V$ and search for a clique containing for each $i \in [1, k]$ exactly one vertex of $V_i$.

Hence, the only additional difficulty for the reduction is to ensure that for each $i \in [1, k]$, exactly one vertex of $V_i$ is selected. To this end, we introduce new characters for each vertex of $V$ that ensure that a binary $X$-tree $T'$ is not improving over $T$, if for some $i \in [1, k]$, there are distinct vertices $v$ and $w$ of $V_i$, such that $T'(X_v)$ is an in-rooting of $T_v$ and $T'(X_w)$ is an in-rooting of $T_w$. We do this by introducing new characters $S_{\text{mcc}}$ and increasing the values of $\beta$ and $\alpha$ to ensure that, again, each binary $X$-tree that improves over $T$ is well-rooted.

First, we replace the taxon $x^*$ by two new taxa $x_0^*$ and $x_1^*$. Abusing notation, we call the resulting set of taxa $X$ as well. In the initial binary $X$-tree $T$, we make both these new taxa adjacent with the new internal vertex $x^*$. For each already defined character $c$, we set $c(x_0^*) := c(x_1^*) := 1$. Hence, the score of each optimal extension of each already defined character $c$ remains the same. Next, we describe the new characters of $S_{\text{mcc}}$. For each $i \in [1, k]$ and each vertex $v \in V_i$, we define a character $c_{v,\text{mcc}}$ as follows. We set $c_{v,\text{mcc}}(x_0^*) := 0$ and $c_{v,\text{mcc}}(x_1^*) := 1$. Moreover, we set $c_{v,\text{mcc}}(x) := 1$ for each taxon $x \in \{\text{in}_v^0, \text{in}_v^1, \overline{\text{in}}_v^0, \overline{\text{in}}_v^1\}$, $c_{v,\text{mcc}}(\overline{\text{out}}_v^0) := c_{v,\text{mcc}}(\overline{\text{out}}_v^1) := 0$, $c_{v,\text{mcc}}(\text{out}_v^0) := 0$, and $c_{v,\text{mcc}}(\text{out}_v^1) := 1$. For each vertex $u \in V_i \setminus \{v\}$, we set $c_{v,\text{mcc}}(x) := 0$ for each taxon $x \in \{\text{in}_u^0, \text{in}_u^1, \overline{\text{in}}_u^0, \overline{\text{in}}_u^1\}$, $c_{v,\text{mcc}}(\overline{\text{out}}_u^0) := c_{v,\text{mcc}}(\overline{\text{out}}_u^1) := 1$, $c_{v,\text{mcc}}(\text{out}_u^0) := 0$, and $c_{v,\text{mcc}}(\text{out}_u^1) := 1$. Finally, for each vertex $w \in V \setminus V_i$, we set $c_{v,\text{mcc}}(x) := 1$ for each taxon $x \in \{\text{in}_w^1, \overline{\text{in}}_w^1, \overline{\text{out}}_w^1, \text{out}_w^1\}$ and we set $c_{v,\text{mcc}}(x) := 0$ for each taxon $x \in \{\text{in}_w^0, \overline{\text{in}}_w^0, \overline{\text{out}}_w^0, \text{out}_w^0\}$. These new characters are depicted in Table 6.2.

The meaning of all relevant character sequences is recalled in Table 6.3.

Let $\gamma := 2 \cdot |X| \cdot (|E| + \binom{k}{2})$. Note that $\gamma$ is larger than $\text{score}_{S_E}(T') + \text{score}_{S_{\text{mal}}}(T')$ of any binary $X$-tree $T'$, since such a tree $T'$ contains less than $2 \cdot |X|$ edges and $|S_E| + |S_{\text{mal}}| = |E| + \binom{k}{2} - 1$. We now add for each vertex $v \in V$ a sequence $S_{v,\text{mcc}}$ of $\gamma$ copies of $c_{v,\text{mcc}}$ to $S$. Let $S_{\text{mcc}}$ be the combined sequence of the sequences $S_{v,\text{mcc}}$ for all vertices $v \in V$. Moreover, include the sequence $S_{\text{mcc}}$ in the sequence $S_G$.

**Table 6.3:** An overview over the different character sequences and the properties these character sequences ensure for any better binary $X$-tree $T'$. Here, $S_V$ denotes the combined sequence of the character sequences $S_v$ for each vertex $v \in V$. The last column indicates the improvement of $T'$ over $T$ with respect to the individual character sequences.

| Characters $S$ | Purpose | Improvement |
|---|---|---|
| $S_R$ | Ensures that for each vertex $v \in V$, $T'(X_v)$ is a pendant subtree of $T'$. | 0 |
| $S_V$ | Ensures that $T'$ is well-rooted. | 0 |
| $S_E$ | Ensures that there are many edges in $G$ between the selected vertices in $T'$. | $\binom{k}{2}$ |
| $S_{\text{mal}}$ | Acts as a hurdle for the required number of edges between selected vertices in $T'$ in $G$. | $-\binom{k}{2}+1$ |
| $S_{\text{mcc}}$ | Ensures that at most one vertex per color class is selected in $T'$. | 0 |

Additionally, we set the value of $\beta$ to $2 \cdot |X| \cdot (|E| + \binom{k}{2} + |S_{\text{mcc}}|)$ and the value of $\alpha$ to $2 \cdot |X| \cdot |S_G|$. Note that $\beta$ is larger than $\text{score}_{S_E}(T') + \text{score}_{S_{\text{mal}}}(T')$ of any binary $X$-tree $T'$, since such a tree $T'$ contains less than $2 \cdot |X|$ edges and $|S_E| + |S_{\text{mal}}| + |S_{\text{mcc}}| = |E| + \binom{k}{2} - 1 + |V| \cdot \gamma$. Similarly, $\alpha$ is larger than $\text{score}_{S_G}(T')$ of any binary $X$-tree $T'$, since such a tree $T'$ contains less than $2 \cdot |X|$ edges.

Intuitively, the large value $\alpha$, again, ensures that only split-consistent binary $X$-trees may improve over $T$. Moreover, the large value of $\beta$ further ensures that only well-rooted binary $X$-trees may improve over $T$. With the newly added characters of $S_{\text{mcc}}$ we now want to further ensure that only well-rooted binary $X$-trees $T'$ may improve over $T$, where for each $i \in [1, k]$, there is at most one vertex $v \in V_i$, such that $T'(X_v)$ is an in-rooting of $v$. We show that this is true by analyzing the score of $S_{\text{mcc}}$ for well-rooted binary $X$-trees.

**Claim 12.** Let $T'$ be a well-rooted binary $X$-tree and let $v$ be a vertex of $V_i$.

a) If $T'(X_v)$ is an out-rooting of $T_v$, then $\text{score}_{c_{v,\text{mcc}}}(T') = 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 1$.

b) If $T'(X_v)$ is an in-rooting of $T_v$ and for each vertex $w \in V_i \setminus \{v\}$, $T'(X_w)$ is an out-rooting of $T_w$, then $\text{score}_{c_{v,\text{mcc}}}(T') = 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 1$.

c) If $T'(X_v)$ is an in-rooting of $T_v$ and for some vertex $w \in V_i \setminus \{v\}$, $T'(X_w)$ is an in-rooting of $T_w$, then $\text{score}_{c_{v,\text{mcc}}}(T') \geq 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 2$.

*Proof of Claim.* For each vertex $w$ of $V$, let $T'_w := T'(X_w)$. Let $V_{\text{in}}$ be those vertices $w$ of $V$, where $T'_w$ is an in-rooting of $T_w$ and let $V_{\text{out}} = V \setminus V_{\text{in}}$ be those vertices $w$ of $V$, where $T'_w$ is an out-rooting of $T_w$. For each vertex $w \in V_{\text{in}}$, let $r_w^{\text{in}}$ be the name of the pseudo-root of $T'_w$, let $r_w^{\text{mid}}$ and $\text{in}_w$ be the neighbors of $r_w^{\text{in}}$, and let $r_w^{\text{out}}$ and $\overline{\text{in}}_w$ be the neighbors of $r_w^{\text{mid}}$. Analogously, for each vertex $w \in V_{\text{out}}$, let $r_w^{\text{out}}$ be the name of the pseudo-root of $T'_w$, let $r_w^{\text{mid}}$ and $\text{out}_w$ be the neighbors of $r_w^{\text{out}}$, and let $r_w^{\text{in}}$ and $\overline{\text{out}}_w$ be the neighbors of $r_w^{\text{mid}}$. Recall that since $T'$ is well-rooted, for each vertex $w \in V$, $\text{in}_w$ is adjacent to both $\text{in}_w^0$ and $\text{in}_w^1$, $\overline{\text{in}}_w$ is adjacent to both $\overline{\text{in}}_w^0$ and $\overline{\text{in}}_w^1$, $\overline{\text{out}}_w$ is adjacent to both $\overline{\text{out}}_w^0$ and $\overline{\text{out}}_w^1$, and $\text{out}_w$ is adjacent to both $\text{out}_w^0$ and $\text{out}_w^1$.

In all three cases, to prove the lower bound for $\text{score}_{c_{v,\text{mcc}}}(T')$, we present a collection of pairwise edge-disjoint paths in $T'$, for which each path contains at least one mutation edge.

For each vertex $w \in V$, let $P_{\text{out}_w}$ be the unique path between $\text{out}_w^0$ and $\text{out}_w^1$ in $T'$. Moreover, for each vertex $w \in V \setminus V_i$,

- let $P_{\text{in}_w}$ be the unique path between $\text{in}_w^0$ and $\text{in}_w^1$ in $T'$,
- let $P_{\overline{\text{in}}_w}$ be the unique path between $\overline{\text{in}}_w^0$ and $\overline{\text{in}}_w^1$ in $T'$, and
- let $P_{\overline{\text{out}}_w}$ be the unique path between $\overline{\text{out}}_w^0$ and $\overline{\text{out}}_w^1$ in $T'$.

Additionally, let $P_{x^*}$ be the unique path between $x_0^*$ and $x_1^*$ in $T'$. Let $\mathcal{P}$ be the collection of all these paths, that is,

$$\mathcal{P} := \{P_{x^*}\} \cup \bigcup_{w \in V} \{P_{\text{out}_w}\} \cup \bigcup_{w \in V \setminus V_i} \{P_{\text{in}_w}, P_{\overline{\text{in}}_w}, P_{\overline{\text{out}}_w}\}.$$

Note that each path of $\mathcal{P}$ contains exactly two edges and all paths of $\mathcal{P}$ are pairwise edge-disjoint.

Moreover, for each vertex $w \in V_i$, let $P_{\text{mid},w}$ be the unique path between $\overline{\text{in}}_w^1$ and $\overline{\text{out}}_w^1$ in $T'$ and let $\mathcal{P}_{V_i}$ denote the collection of all these paths, that is, $\mathcal{P}_{V_i} := \{P_{\text{mid},w} \mid w \in V_i\}$. Note that each path of $\mathcal{P}_{V_i}$ is edge-disjoint with each path of $\mathcal{P}$ and all paths of $\mathcal{P}_{V_i}$ are pairwise edge-disjoint.

Let $P'$ be a path in $\mathcal{P} \cup \mathcal{P}_{V_i}$ and let $w^0$ and $w^1$ be the terminals of $P'$. Since by definition $c_{v,\text{mcc}}(w^0) \neq c_{v,\text{mcc}}(w^1)$, for each extension $c^*_{v,\text{mcc}}$ of $c_{v,\text{mcc}}$ in $T'$, at least one edge of $P'$ is a mutation edge of $c^*_{v,\text{mcc}}$. Consequently, for each extension $c^*_{v,\text{mcc}}$ of $c_{v,\text{mcc}}$ in $T'$, $\text{score}_{c^*_{v,\text{mcc}}}(T') \geq |\mathcal{P} \cup \mathcal{P}_{V_i}| = 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 1$ which implies $\text{score}_{c_{v,\text{mcc}}}(T') \geq 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 1$.

Hence, to prove the claim, for statements a) and b), it remains to show that there is an extension $c^*_{v,\mathrm{mcc}}$ of $c_{v,\mathrm{mcc}}$ in $T'$ with $\mathrm{score}_{c^*_e}(T') \leq 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 1$, and for statement c) it is sufficient to present a path $P$ in $T'$ which is edge-disjoint with each path of $\mathcal{P} \cup \mathcal{P}_{V_i}$ and containing at least one mutation edge.

We start by showing statement c). Hence, we know that $T'_v$ is an in-rooting of $T_v$ and that there is some vertex $w \in V_i \setminus \{v\}$, such that $T'_w$ is an in-rooting of $T_w$. Let $P$ be the unique path between $\mathrm{in}^0_v$ and $\mathrm{in}^0_w$ in $T'$. By definition, $c_{v,\mathrm{mcc}}(\mathrm{in}^0_v) = 1 \neq 0 = c_{v,\mathrm{mcc}}(\mathrm{in}^0_w)$. Hence, for each extension of $c_{v,\mathrm{mcc}}$ in $T'$, $P$ contains at least one mutation edge. To show that for each extension $c^*_{v,\mathrm{mcc}}$ of $c_{v,\mathrm{mcc}}$ in $T'$, $\mathrm{score}_{c^*_e}(T') \geq 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 2$, it thus remains to show that $P$ is edge-disjoint with each path of $\mathcal{P} \cup \mathcal{P}_{V_i}$. By that fact that $T'_v$ is an in-rooting of $T_v$ and $T'_w$ is an in-rooting of $T_w$, this is the case.

Next, we show statement a). Hence, we assume that $T'_v$ is an out-rooting of $T_v$. We define an extension $c^*_{v,\mathrm{mcc}}$ of $c_{v,\mathrm{mcc}}$ in $T'$ with $\mathrm{score}_{c^*_e}(T') \leq 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 1$. This extension is illustrated in Figure 6.9. We set $c^*_{v,\mathrm{mcc}}(\mathrm{in}_v) := c^*_{v,\mathrm{mcc}}(\overline{\mathrm{in}}_v) := c^*_{v,\mathrm{mcc}}(r^{\mathrm{in}}_v) := 1$. Moreover, for each vertex $w \in V_i \setminus \{v\}$, we set $c^*_{v,\mathrm{mcc}}(\overline{\mathrm{out}}_w) = 1$. For each remaining internal vertex $v'$ of $T'$, we set $c^*_{v,\mathrm{mcc}}(v') := 0$. Hence, the edge set

$$\begin{aligned}\{\{r^{\mathrm{in}}_v, r^{\mathrm{mid}}_v\}, \{x^*_1, x^*\}\} &\cup \{\{\overline{\mathrm{in}}_w, r^{\mathrm{out}}_w\} \mid w \in V_i \cap V_{\mathrm{in}}\} \\ &\cup \{\{\overline{\mathrm{in}}_w, r^{\mathrm{mid}}_w\} \mid w \in V_i \cap V_{\mathrm{out}}, w \neq v\} \\ &\cup \{\{\mathrm{out}^1_w, \mathrm{out}_w\} \mid w \in V\} \\ &\cup \{\{\mathrm{in}^1_w, \mathrm{in}_w\}, \{\overline{\mathrm{in}}^1_w, \overline{\mathrm{in}}_w\}, \{\overline{\mathrm{out}}^1_w, \overline{\mathrm{out}}_w\} \mid w \in V \setminus V_i\}\end{aligned}$$

contains the mutation edges of $c^*_{v,\mathrm{mcc}}$ in $T'$. Consequently, $\mathrm{score}_{c^*_{v,\mathrm{mcc}}}(T') = 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 1$ which implies $\mathrm{score}_{c_{v,\mathrm{mcc}}}(T') = 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 1$ by the above lower bound for $\mathrm{score}_{c_{v,\mathrm{mcc}}}(T')$.

Finally, we show statement b). Hence, we assume that $T'_v$ is an in-rooting of $T_v$ and that for each vertex $w \in V_i \setminus \{v\}$, $T'_w$ is an out-rooting of $T_w$. We define an extension $c^*_{v,\mathrm{mcc}}$ of $c_{v,\mathrm{mcc}}$ in $T'$ with $\mathrm{score}_{c^*_e}(T') \leq 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 1$. This extension is illustrated in Figure 6.9. We set $c^*_{v,\mathrm{mcc}}(\overline{\mathrm{out}}_v) := 0$. Moreover, for each vertex $w \in V_i \setminus \{v\}$, we set $c^*_{v,\mathrm{mcc}}(\mathrm{in}_v) := c^*_{v,\mathrm{mcc}}(\overline{\mathrm{in}}_v) := c^*_{v,\mathrm{mcc}}(r^{\mathrm{in}}_v) := 0$. For each remaining internal vertex $v'$ of $T'$, we set $c^*_{v,\mathrm{mcc}}(v') := 1$. Hence, the edge set

$$\begin{aligned}\{\{\overline{\mathrm{out}}_v, r^{\mathrm{out}}_v\}, \{x^*_0, x^*\}\} &\cup \{\{r^{\mathrm{in}}_w, r^{\mathrm{mid}}_w\} \mid w \in V_i \setminus \{v\}\} \cup \{\{\mathrm{out}^0_w, \mathrm{out}_w\} \mid w \in V\} \\ &\cup \{\{\mathrm{in}^0_w, \mathrm{in}_w\}, \{\overline{\mathrm{in}}^0_w, \overline{\mathrm{in}}_w\}, \{\overline{\mathrm{out}}^0_w, \overline{\mathrm{out}}_w\} \mid w \in V \setminus V_i\}\end{aligned}$$

contains the mutation edges of $c^*_{v,\mathrm{mcc}}$ in $T'$. Consequently, $\mathrm{score}_{c^*_{v,\mathrm{mcc}}}(T') = 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 1$ which implies $\mathrm{score}_{c_{v,\mathrm{mcc}}}(T') = 2 \cdot |V_i| + 4 \cdot |V \setminus V_i| + 1$ by the above lower bound for $\mathrm{score}_{c_{v,\mathrm{mcc}}}(T')$. ■

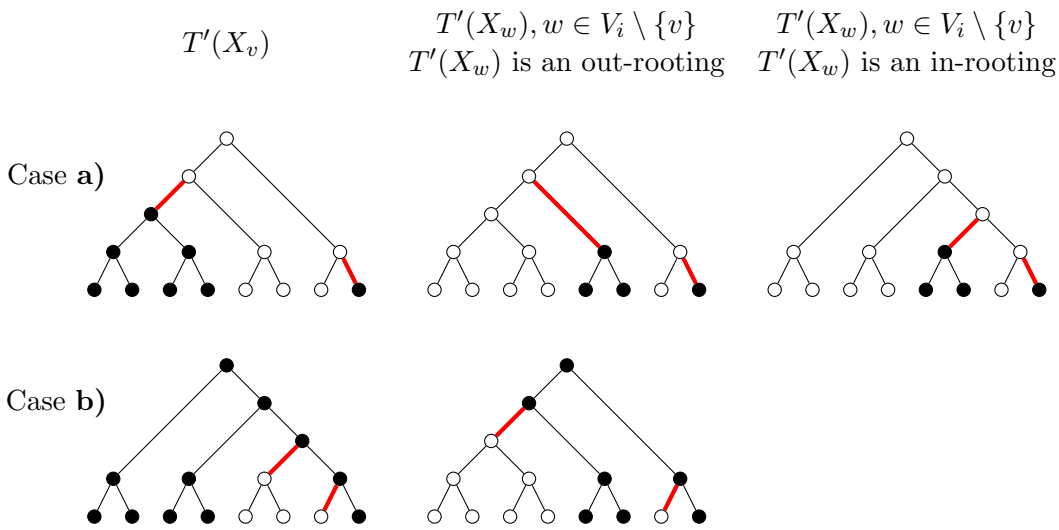



**Figure 6.9:** An optimal extension of $c_{v,\text{mcc}}$ for the cases a) and b) of Claim 12 for any well-rooted binary $X$-tree $T'$, where $v$ is a vertex of $V_i$. White vertices receive state 0 and black vertices receive state 1 under these extensions. The mutation edges inside the depicted subtrees are highlighted in red. Note that each depicted subtree contains exactly two mutation edges under these extensions. All non-depicted internal vertices of $T'$ receive label 0 in case a) and label 1 in case b). Hence, the depicted subtrees include all mutation edges that are not incident with a leaf of $T'$. Note that in case b), there is no vertex $w \in V_i \setminus \{v\}$, such that $T'(X_w)$ is an in-rooting of $T_w$.

Since $T$ is well-rooted and for each vertex $w \in V$, $T_w$ is an out-rooting of $T_w$, this implies that $T$ achieves the minimum possible score for all characters of $S_{\text{mcc}}$ of any well-rooted binary $X$-tree. Hence, a binary $X$-tree $T'$ can only improve over $T$ if $T'$ is well-rooted and for each $i \in [1, k]$, there is at most one vertex $v \in V_i$, such that $T'(X_v)$ is an in-rooting of $X_v$. This holds, since i) $T$ achieves the minimum possible total score for all characters of $S$ besides $S_{\text{mal}}$ and $S_E$, ii) the value of $\gamma$ is larger than the total possible score of the characters of $S_{\text{mal}}$ and $S_E$ of any binary $X$-tree, and iii) $S$ contains $\gamma$ copies of $c_{v,\text{mcc}}$ for each vertex $v \in V$.

Note that this further implies that if there is a binary $X$-tree $T'$ improving over $T$, then $T'$ is well-rooted and there is a set $K \subseteq V$ of size at most $k$ such that for each vertex $v \in K$, $T'(X_v)$ is an in-rooting of $T_v$. Due to Lemma 6.17, this implies that each binary $X$-tree $T'$ improving over $T$ has distance at most $k'$ with $T$ with respect to the considered distance measure $d$. This then implies that both the permissive and the strict version on the constructed instance of $d$-LS Maximum Parsimony are equivalent. Hence, to show that the hardness results also transfer to the permissive

version of $d$-LS Maximum Parsimony, it thus remains to show that this modified instance of $d$-LS Maximum Parsimony is still a yes-instance if and only if the initial instance of Clique is a yes-instance. This follows from the proof of Lemma 6.18. □

## 6.5 Essentially Tight Brute-Force Algorithms

We now show that simple brute-force algorithms for $d$-LS Maximum Parsimony for each distance measure $d \in \{d_{\text{NNI}}, d_{\text{ECR}}, d_{\text{SPR}}, d_{\text{TBR}}\}$ essentially match the lower bounds shown in Theorem 6.9. First, consider the distance measures $d_{\text{NNI}}$, $d_{\text{SPR}}$, and $d_{\text{TBR}}$.

**Observation 6.20.** *Let $T$ be a binary $X$-tree, let $d \in \{d_{\text{NNI}}, d_{\text{SPR}}, d_{\text{TBR}}\}$ be a distance measure, and let $k$ be an integer. One can enumerate all binary $X$-trees $T'$ with $d(T, T') \leq k$ in $|X|^{\mathcal{O}(k)}$ time.*

The correctness of Observation 6.20 can be seen as follows: there are $|X|^{\mathcal{O}(1)}$ many binary $X$-trees $T'$ such that $d(T, T') = 1$ [3], all these trees can be enumerated in $|X|^{\mathcal{O}(1)}$ time, and for each binary $X$-tree $T'$ with $d(T, T') > 0$, there is a binary $X$-tree $\widehat{T}$ with $d(\widehat{T}, T') = 1$ and $d(T, T') = d(T, \widehat{T}) + 1$.

**Enumerating trees of small sECR-distance.** Furthermore, we may enumerate all binary $X$-trees $T'$ with $d_{\text{sECR}}(T, T') \leq k$ as follows: First, we enumerate all subtrees of $T$ with at most $k$ edges. Second, for each connected subtree $T_s$ of $T$ with at most $k$ edges, we enumerate all binary refinements of $T$ after contracting all edges of $T_s$ in $T$. In Lemma 6.21, we show that the first step can be done in $\mathcal{O}(4^k \cdot k^{-0.5} \cdot |X|)$ time. In Lemma 6.22, we show that both steps can be performed in $\mathcal{O}((2k+1)!! \cdot 4^k \cdot k\sqrt{k} \cdot |X|^2)$ time where $(2k+1)!! := 1 \cdot 3 \cdot \ldots \cdot (2k+1)$.

**Lemma 6.21.** *For every binary $X$-tree $T$ and every integer $k$, all connected subtrees of $T$ with at most $k$ edges can be enumerated in $\mathcal{O}(4^k \cdot k^{-0.5} \cdot |X|)$ time.*

*Proof.* Let $n := |X|$. Let $\mathcal{C}_\ell$ be the $\ell$th Catalan number. For any fixed vertex $v$ of $T$, there are at most $\frac{3}{2\ell+1}\binom{2\ell+1}{\ell-1} = \mathcal{C}_{\ell+1} - \mathcal{C}_\ell$ subtrees of $T$ with $\ell$ vertices that

contain $v$ [123]. Thus, there are at most

$$\begin{aligned} n \cdot (k+1) \cdot \frac{3}{2k+3} \cdot \binom{2k+3}{k} &= n \cdot \frac{3 \cdot (2k+2)!}{(k+2)! \cdot k!} \cdot \frac{k+1}{k+3} \\ &\leq n \cdot \frac{3 \cdot (2k+2)!}{(k+1)! \cdot (k+1)!} = n \cdot 3 \cdot \binom{2k+2}{k+1} = n \cdot (k+1) \cdot \mathcal{C}_{k+1} \\ &\leq n \cdot (k+1) \cdot \frac{4^{k+1}}{(k+2) \cdot \sqrt{\pi \cdot (k+2)}} \in \mathcal{O}\left(4^k \cdot k^{-0.5} \cdot n\right) \end{aligned}$$

subtrees of $T$ that contain at most $k$ edges. Note that the last two inequalities holds, since for each integer $\ell$, $\mathcal{C}_\ell = \frac{3}{\ell+1}\binom{2\ell}{\ell} \leq \frac{4^\ell}{(\ell+1)\cdot\sqrt{\pi\ell}}$.

For a given integer $\ell$ and a tree $T$, all subtrees of $T$ with at most $\ell$ edges can be enumerated in $\mathcal{O}(\alpha)$ time, where $\alpha$ is the number of subtrees of $T$ with $\ell$ edges [172]. It follows that all subtrees of $T$ that contain at most $k$ edges can be enumerated in $\mathcal{O}(4^k \cdot k^{-0.5} \cdot n)$ time. □

Based on the algorithm behind Lemma 6.21, we now present an enumeration algorithm for all binary $X$-trees that have sECR-distance at most $k$ with some given binary $X$-tree $T$.

**Lemma 6.22.** *For a given binary $X$-tree $T$ and an integer $k$, there are $\mathcal{O}((2k+1)!! \cdot 4^k \cdot k^{-0.5} \cdot |X|)$ binary $X$-trees $T'$ with $d_{\mathrm{sECR}}(T, T') \leq k$. Moreover, all these binary $X$-tree can be enumerated in $\mathcal{O}((2k+1)!! \cdot 4^k \cdot k \cdot \sqrt{k} \cdot |X|^2)$ time.*

*Proof.* The algorithm works as follows: Enumerate all subtrees of $T$ with at most $k$ edges that do not contain a pendant edge of $T$. For each of these subtrees $T_s = (V_s, E_s)$, compute the neighborhood $V^*$ of the vertices of $V_s$ in $T$ outside of $T_s$. Let $T_C$ be the tree obtained from contracting the whole subtree $T_s$ in $T$ into a single vertex $\bar{v}$. Observe that the neighborhood of $\bar{v}$ in $T_C$ is exactly $V^*$. Iterate over each binary refinement of $T_C$ and return the resulting $X$-trees.

We show that the algorithm is correct. By definition of sECR operations, the fact that we considered all subtrees $T_s$ of $T$ with at most $k$ edges, and the fact that we considered for each such subtree $T_s$ all binary refinements of the tree obtained from $T$ by contracting the edges of $T_s$, the algorithm enumerates exactly those binary $X$-trees $T'$ with $d_{\mathrm{sECR}}(T, T') \leq k$.

It remains to show that the algorithm runs in the stated running time. By Lemma 6.21, the $\mathcal{O}(4^k \cdot k^{-0.5} \cdot |X|)$ subtrees of $T$ with at most $k$ edges can be enumerated in $\mathcal{O}(4^k \cdot k^{-0.5} \cdot |X|)$ time. For each subtree $T_s$ of $T$ with at most $k$ edges, the set of vertices $V^*$ and the tree $T_C$ can be computed in $\mathcal{O}(|X|)$ time.

Note that $T_C(V^*)$ is a star with $|V^*| = k + 3$ leaves, and that each binary refinement of $T_C(V^*)$ is a binary $V^*$-tree and vice versa. Hence, there are $(2k + 1)!!$ possible binary refinements of $T_C(V^*)$ [149, 152] and all these binary refinements of $T_C(V^*)$ can be enumerated in $\mathcal{O}((2k + 1)!! \cdot k^2)$ time [28, 30]. Overall, for a given binary $X$-tree $T$ the $\mathcal{O}((2k+1)!! \cdot 4^k \cdot k^{-0.5} \cdot |X|)$ binary $X$-trees $T'$ with $d_{\mathrm{sECR}}(T, T') \leq k$ can be enumerated in $\mathcal{O}((2k + 1)!! \cdot 4^k \cdot k\sqrt{k} \cdot |X|^2)$ time. □

Hence, we obtain the following due to the fact that the parsimony score of a given $X$-tree can be computed in $\mathcal{O}(|X| \cdot |S|)$ time [58].

**Theorem 6.23.** *$d_{\mathrm{sECR}}$-LS Maximum Parsimony can be solved in $\mathcal{O}((2k + 1)!! \cdot 4^k \cdot k\sqrt{k} \cdot |X|^2 \cdot |S|) = 2^{\mathcal{O}(k \cdot \log k)} \cdot |X|^2 \cdot |S|$ time.*

**Enumerating trees of small ECR-distance.** Finally, we describe how to enumerate all binary $X$-trees $T'$ with $d_{\mathrm{ECR}}(T, T') \leq k$.

**Lemma 6.24.** *Let $T$ be a binary $X$-tree and let $k$ be an integer. One can enumerate all binary $X$-trees $T'$ with $d_{\mathrm{ECR}}(T, T') \leq k$ in $|X|^{\mathcal{O}(k)}$ time.*

*Proof.* We show this statement by induction over $k$.

**Base case.** Consider $k = 0$. Hence, $T$ is the unique binary $X$-tree $T'$ that fulfills $d_{\mathrm{ECR}}(T, T') = 0$. Hence, the statement holds for $k = 0$.

**Inductive step.** For the inductive step, suppose that for each binary $X$-tree $\widetilde{T}$ and for each $k' < k$, one can compute all binary $X$-trees $T'$ with $d_{\mathrm{ECR}}(\widetilde{T}, T') \leq k'$ in $|X|^{\mathcal{O}(k')}$ time. Note that this implies that for each $k' < k$ there are $|X|^{\mathcal{O}(k')}$ binary $X$-trees $T'$ with $d_{\mathrm{ECR}}(\widetilde{T}, T') = k'$. For each $i < k$, let $\mathcal{T}_i$ be the collection of all binary $X$-trees $\widetilde{T}$ with $d_{\mathrm{ECR}}(T, \widetilde{T}) = i$ and let $\mathcal{T}_{<k}$ be the collection of all binary $X$-trees $\widetilde{T}$ with $d_{\mathrm{ECR}}(T, \widetilde{T}) < k$, that is, $\mathcal{T}_{<k} = \cup_{i=0}^{k-1} \mathcal{T}_i$. Moreover, let $\mathcal{T}_{\mathrm{sECR}}$ be the collection of all binary $X$-trees $\widetilde{T}$ with $d_{\mathrm{sECR}}(T, \widetilde{T}) = k$. Note that $\mathcal{T}_{<k}$ can be computed in $|X|^{\mathcal{O}(k-1)}$ time and $\mathcal{T}_{\mathrm{sECR}}$ can be computed in $k^{\mathcal{O}(k)} \cdot |X|^{\mathcal{O}(1)}$ time due to Lemma 6.22. Let

$$\mathcal{T}'_k := \mathcal{T}_{\mathrm{sECR}} \cup \bigcup_{i=1}^{k-1} \bigcup_{\widetilde{T} \in \mathcal{T}_i} \{T' \mid d_{\mathrm{ECR}}(\widetilde{T}, T') \leq k - i\}.$$

Recall that by the induction hypothesis, for each $i < k$, $\mathcal{T}_i$ has size $|X|^{\mathcal{O}(i)}$ and for each binary $X$-tree $\widetilde{T} \in \mathcal{T}_i$ the collection $\{T' \mid d_{\mathrm{ECR}}(\widetilde{T}, T') \leq k - i\}$ can be computed in $|X|^{\mathcal{O}(k-i)}$ time. Hence, $\mathcal{T}'_k$ can be computed in $|X|^{\mathcal{O}(k)}$ time. We set $\mathcal{T} := \mathcal{T}'_k \cup \mathcal{T}_{<k}$ and show that $\mathcal{T}$ contains exactly the binary $X$-trees $T'$ with $d_{\mathrm{ECR}}(T, T') \leq k$.

Assume towards a contradiction that this is not the case.

**Case 1:** There is a binary $X$-tree $T'$ with $d_{\mathrm{ECR}}(T,T') \leq k$ such that $T'$ is not in $\mathcal{T}$. By definition, $\mathcal{T}_{<k}$ contains all binary $X$-trees $\widetilde{T}$ with $d_{\mathrm{ECR}}(T,\widetilde{T}) < k$. Consequently, $d_{\mathrm{ECR}}(T,T') = k$. Hence, due to Observation 6.5, there is a binary $X$-tree $\widetilde{T}$ with $d_{\mathrm{sECR}}(\widetilde{T},T') > 0$ such that $d_{\mathrm{ECR}}(T,T') = d_{\mathrm{ECR}}(T,\widetilde{T}) + d_{\mathrm{sECR}}(\widetilde{T},T')$. Let $i := d_{\mathrm{ECR}}(T,\widetilde{T})$.

Note that $i \leq k-1$. If $i = 0$, then $T$ is isomorphic to $\widetilde{T}$ and thus $d_{\mathrm{sECR}}(T,T') = d_{\mathrm{sECR}}(\widetilde{T},T') = k$. Hence, $T'$ is contained in $\mathcal{T}_{\mathrm{sECR}}$, a contradiction. Otherwise, if $i > 0$, then $\widetilde{T}$ is contained in $\mathcal{T}_i$. Moreover, since $d_{\mathrm{sECR}}(\widetilde{T},T') = d_{\mathrm{ECR}}(T,T') - d_{\mathrm{ECR}}(T,\widetilde{T}) = k - i$ and $d_{\mathrm{sECR}}(\widetilde{T},T') \geq d_{\mathrm{ECR}}(\widetilde{T},T')$, we have $d_{\mathrm{ECR}}(\widetilde{T},T') \leq k - i$ which implies that $T'$ is contained in $\mathcal{T}$, a contradiction.

**Case 2:** There is a binary $X$-tree $T'$ with $d_{\mathrm{ECR}}(T,T') > k$ such that $T'$ is contained in $\mathcal{T}$. Hence, $T'$ is contained in $\mathcal{T}'_k \setminus \mathcal{T}_{\mathrm{sECR}}$. That is, there is some $i$ with $1 \leq i \leq k$ and a binary $X$-tree $\widetilde{T}$ in $\mathcal{T}_i$ such that $d_{\mathrm{ECR}}(\widetilde{T},T') \leq k - i$. Since $d_{\mathrm{ECR}}$ is a metric, due to the triangle inequality, $d_{\mathrm{ECR}}(T,T') \leq d_{\mathrm{ECR}}(T,\widetilde{T}) + d_{\mathrm{ECR}}(\widetilde{T},T') \leq k$, a contradiction.

Since $\mathcal{T}$ can be computed in $|X|^{\mathcal{O}(k)}$ time, the statement holds. □

Since for any two binary $X$-trees $T$ and $T'$, $2 \cdot d_{\mathrm{ECR}}(T,T') = d_{\mathrm{RF}}(T,T')$, we also obtain the following result for enumerating all binary $X$-trees of small Robinson-Foulds-distance.

**Corollary 6.25.** *Let $T$ be a binary $X$-tree and let $k$ be an integer. One can enumerate all binary $X$-trees $T'$ with $d_{\mathrm{RF}}(T,T') \leq k$ in $|X|^{\mathcal{O}(k)}$ time.*

Due to Observation 6.20, Lemma 6.24, and Corollary 6.25, we conclude the following.

**Theorem 6.26.** *For each distance measure $d \in \{d_{\mathrm{NNI}}, d_{\mathrm{ECR}}, d_{\mathrm{RF}}, d_{\mathrm{SPR}}, d_{\mathrm{TBR}}\}$, $d$-*LS Maximum Parsimony *can be solved in $|X|^{\mathcal{O}(k)} \cdot |S|$ time.*

*Proof.* Let $d$ be a distance measure of $\{d_{\mathrm{NNI}}, d_{\mathrm{ECR}}, d_{\mathrm{RF}}, d_{\mathrm{SPR}}, d_{\mathrm{TBR}}\}$ and let $I = (X,T,d,S,k)$ be an instance of $d$-LS Maximum Parsimony. Due to Observation 6.20, Lemma 6.24, and Corollary 6.25, we can compute the collection $\mathcal{T}$ of all binary $X$-trees $T'$ with $d(T,T') \leq k$ in $|X|^{\mathcal{O}(k)}$ time. We compute $\mathrm{score}_S(T')$ for each binary $X$-tree $T' \in \mathcal{T}$ and answer yes if and only if there is a binary $X$-tree $T' \in \mathcal{T}$ with $\mathrm{score}_S(T') < \mathrm{score}_S(T)$. By definition of $\mathcal{T}$, the algorithm is correct. Moreover, the algorithm runs in $|X|^{\mathcal{O}(k)} \cdot |S|$ time, since $\mathcal{T}$ has size at most $|X|^{\mathcal{O}(k)}$ and for each binary $X$-tree $T'$ of $\mathcal{T}$, one can compute $\mathrm{score}_S(T')$ in $\mathcal{O}(|X| \cdot |S|)$ time [58]. □

## 6.6 Concluding Remarks

In this chapter, we analyzed the parameterized complexity of searching for a better solution in the scalable $k$-neighborhood of a phylogenetic tree $T$ for Maximum Parsimony with respect to the distance measures $d_{\text{NNI}}$, $d_{\text{SPR}}$, $d_{\text{TBR}}$, $d_{\text{ECR}}$, $d_{\text{RF}}$, and $d_{\text{sECR}}$. For the first five distance measures, we showed that finding algorithms that run significantly faster than a trivial brute-force algorithm with running time $|I|^{\mathcal{O}(k)}$ are impossible, unless the ETH fails. This implies that there is a large algorithmic difference between finding a better or even a best solution of distance at most $k$ and the question whether two given phylogenetic trees have distance at most $k$. This is the case, since the latter task was shown to be solvable in $f(k) \cdot |I|^{\mathcal{O}(1)}$ time for the distance measures $d_{\text{SPR}}$ [174], $d_{\text{TBR}}$ [86], and $d_{\text{RF}}$ [145] (and thus also for $d_{\text{ECR}}$). With respect to the last distance measure $d_{\text{sECR}}$, we developed an FPT-algorithm that searches the $k$-sECR neighborhood in $k^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time.

**Open questions.** A clear goal for future research would be to improve the running time of the algorithm for the $k$-sECR neighborhood. At first, this seems promising since the current bottleneck is the enumeration of the binary refinements of the tree obtained after contracting $k$ edges. However, an algorithm for $d_{\text{sECR}}$-LS Maximum Parsimony running in $2^{o(k \cdot \log k)} \cdot |I|^{\mathcal{O}(1)}$ time would imply an algorithm for Maximum Parsimony running in $2^{o(|X| \cdot \log |X|)} \cdot |I|^{\mathcal{O}(1)}$ time: when applying the $d_{\text{sECR}}$-LS Maximum Parsimony algorithm with $k := |X|-3$, locally optimal solution are also globally optimal. This is due to the fact that the $d_{\text{sECR}}$-LS Maximum Parsimony-distance is at most $|X| - 3$ between any two binary $X$-trees, since each binary $X$-tree has exactly $|X| - 3$ internal edges. Hence, a more immediate question is whether Maximum Parsimony can be solved in $2^{o(|X| \cdot \log |X|)} \cdot |I|^{\mathcal{O}(1)}$ time.

A further goal would be to find other distance measures for which $d$-LS Maximum Parsimony can be solved in time $f(k) \cdot |I|^{\mathcal{O}(1)}$. An example of a distance measure on phylogenetic trees one might look into is the *quartet distance* [26, 51]. The *quartet topologies* induced by a binary $X$-tree $T$ is the set containing, for each set $X' \subseteq X$ of size exactly 4, the binary $X'$-tree obtained from suppressing each degree-2 vertex in $T(X')$. Then, for two phylogenetic $X$-trees $T$ and $T'$, the quartet distance between $T$ and $T'$ is the number of quartet topologies induced by exactly one of $T$ and $T'$ [51]. In our reductions, the distance between the initial $X$-tree and any improving $X$-tree is $\Omega(n)$. Hence, our presented lower bounds can not be derived directly for the quartet distance. This leaves open whether one can search for a better $X$-tree of quartet distance at most $k$ in time $f(k) \cdot |I|^{\mathcal{O}(1)}$.

Furthermore, it is open whether better running times are possible when searching

the neighborhood not for a better tree, but for a perfect phylogeny, that is, for a tree where for each character the parsimony score equals the number of character states minus one. A related question was considered by Szeider [161] in context of local search for SAT. For $k$-Flip Sat they considered as input a formula $F$ in CNF together with an assignment $\tau$ of the variables of $F$ and asked whether there is an assignment for $F$ that satisfies all clauses and that can be obtained from $\tau$ by flipping the truth values of at most $k$ variables. In contrast to $k$-Flip Max Sat (the problem, where one only asks for an assignment that satisfies more clauses than $\tau$), they showed that $k$-Flip Sat is FPT with respect to $k$ plus the maximum number of literals per clause [161]. This shows that one might be able to exploit the structural properties of "perfect" solutions algorithmically, when only searching for such "perfect" solutions.

A different question, more related to the approaches we aimed for in Chapters 3–5, is to analyze whether we can find additional structural parameters $\ell$ for which we can solve $d$-LS Maximum Parsimony in $\ell^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time for any $d \in \{d_{\mathrm{NNI}}, d_{\mathrm{SPR}}, d_{\mathrm{TBR}}, d_{\mathrm{ECR}}, d_{\mathrm{RF}}\}$. Note that our algorithms presented in Section 6.5 achieve such running times for $\ell$ being the number of taxa. It would be interesting to analyze whether we can find smaller parameters or even unrelated parameters for which such algorithms are possible. For example, once can consider $\ell$ being the number of taxa one has to remove from $X$ to obtain a perfect phylogeny [76]. This parameter is clearly smaller than $|X|$. Another example is to consider $\ell$ being a parameter upper bounded by the number of characters $|S|$. In a first step, one can aim to find algorithms that run in $|S|^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time. If such algorithms are possible, one can consider as parameter $\ell$ for example the number of *bad characters in* $T$, that is, the characters for which $T$ is not a perfect phylogeny [76].

Finally, one can also ask for the complexity of finding a locally optimal solution with respect to scalable $k$-neighborhood defined over the distance measures $d_{\mathrm{NNI}}$, $d_{\mathrm{SPR}}$, $d_{\mathrm{TBR}}$, and $d_{\mathrm{ECR}}$ for small constant values of $k$. Since each phylogenetic tree contains $\Theta(|X|)$ edges, and each such edge can contribute at most 1 to the parsimony score of each character of $S$, the parsimony score of any phylogenetic tree is upper-bounded by $\mathcal{O}(|X| \cdot |S|)$. Hence, a locally optimal solution can be found in polynomial time, since a better tree of constant distance from a current tree can be found in polynomial time, if it exists (see Theorem 6.26). So, this question is more interesting in the context of a weighted version of Maximum Parsimony. In our opinion, such a weighted version is well-motivated, for example by the following consideration. Based on the fact that a huge bottleneck in real data for many biological problems is the enormously large amount of character data (for example, mammal genomes usually range from $1.4 \cdot 10^9$ to $3.7 \cdot 10^9$ base pairs [142]), considering a version of Maximum

Parsimony with a compressed representation of the characters might be desirable. For example, if two characters $c$ and $c'$ are isomorphic (that is, if there is a bisection $\pi$, such that $c(x) = \pi(c'(x))$ for each taxon $x \in X$), the parsimony scores of these characters are identical with respect to every fixed phylogenetic tree. Hence, instead of storing each character of $S$ explicitly, it suffices to only store all pairwise non-isomorphic characters $S'$ of $S$ and store for each character $c$ of $S'$ some additional weight $\omega_c$ that denotes the number of characters of $S$ which are isomorphic to $c$. These weights $\omega_c$ can then be encoded in binary, which might provide a significant compression of the input size in comparison to the unweighted encoding of Maximum Parsimony. Now, the maximal possible parsimony score of any tree is upper-bounded by $\mathcal{O}(|X| \cdot \sum_{c \in S'} \omega_c)$. Due to the binary encoding of $\omega_c$, this value is not necessarily polynomial in the total input size. Consequently, in such a weighted setting, it is not guaranteed that a locally optimal solution can be found in polynomial time. To analyze whether this is still possible in polynomial time, one might consider analyzing whether this weighted version of Maximum Parsimony is PLS-hard with respect to the $k$-neighborhoods $d_{\text{NNI}}$, $d_{\text{SPR}}$, $d_{\text{TBR}}$, and $d_{\text{ECR}}$ for small constant values of $k$.

# Chapter 7

# The Complexity of Finding $k$-Swap-Optimal Solutions for Subset Optimization Problems

In this chapter, we consider Question 2 which asks about the total running time to find a locally optimal solution. For a large class of subset-weight optimization problems, we analyze the complexity of finding locally optimal solutions for instances of these problems with respect to the $k$-swap neighborhood for constant values of $k$. For example, we analyze this question for Weighted Independent Set and the more general Weighted $\Pi$ Subgraph.

Recall that the complexity class PLS (formally defined in Section 2.7) introduced by Johnson et al. [95] is designed to study the difficulty of computing locally optimal solutions. This class contains all local search problems for which one can compute some starting solution and search the local neighborhood of a feasible solution $S$ in polynomial time. Thus, the local search problems in PLS are exactly those that have a hill-climbing algorithm where each step only takes polynomial time.

To give evidence that for some local search problems in PLS it may be hard to compute locally optimal solutions, Johnson et al. [95] introduced PLS-reductions and showed that there are PLS-complete problems. These problems are as hard as any problem in PLS and none of them is known to be solvable in polynomial time. Further evidence against polynomial-time algorithms for PLS-complete problems arises from cryptographic assumptions: Daskalakis and Papadimitriou introduced the class CLS [37] which is a subclass of PLS. Now under the existence of one-way permutations and an indistinguishability obfuscation assumption, there are problems in CLS, and thus in PLS, that do not admit polynomial-time algorithms [92].

**Related work.** PLS-completeness has been shown for many local search problems [46, 47, 95, 121, 125, 151, 154]. The initial PLS-complete problem is called Flip, where the input is a Boolean circuit with $m$ input variables and $n$ output variables. The solutions are the bit-vectors of length $m$ and the objective is to find an input that has a lexicographically smallest output. The neighborhood of a vector of the input variables is obtained by flipping one input bit [95]. Another prominent PLS-complete problem is Max Cut with the flip neighborhood, where the neighbors of a partition $(A, B)$ are the partitions that can be obtained by moving one vertex from $A$ to $B$ or vice versa [151]. Max Cut with the flip neighborhood is PLS-complete on graphs with maximum degree 5 [47] and polynomial-time solvable on 3-regular graphs [141]. The complexity on graphs with maximum degree 4 remains open [47]. Johnson et al. [95] already considered the Weighted Independent Set problem and argued that one can show PLS-completeness for a neighborhood that is inspired by the famous Kernighan-Lin algorithm for Max Cut [100]. Moreover, Johnson et al. [95] specifically called for the study of simpler neighborhoods for Weighted Independent Set. Schäffer and Yannakakis [151] argued that Weighted Independent Set is PLS-complete for a 2-step neighborhood which consists of an addition of a vertex $v$ to the independent set $S$ and a removal of all its neighbors from $S$ in the first step and a (maximal) series of improving vertex additions in the second step. Note that the first step is not necessarily improving. Further studies have shown PLS-completeness for Weighted Independent Dominating Set with a $k$-swap neighborhood with constant but unspecified $k$ [101] and for Weighted Max-Π-Subgraph with hereditary properties Π and the above-described 2-step-neighborhood [154]. We are the first to analyze the complexity of Weighted Independent Set with respect to the $k$-swap neighborhoods for $k \leq 3$.

From a practical point of view, local search has been shown to give very good results for WIS [41, 116, 133]. For example, the iterated local search heuristic called ILS-VND [133]. The neighborhood used in this heuristic allows (i) swaps that remove one vertex from the independent set and add up to two vertices to the independent set and (ii) swaps that add one vertex to the independent set and remove all its neighbors from the independent set. Note that this second operation may swap $\Delta(G) + 1$ vertices simultaneously. Later, another heuristic called METAMIS was shown to outperform ILS-VND [41]. METAMIS also has a local search step which allows the swaps of ILS-VND and in addition also the swaps that remove two vertices from the independent set and add a fixed number of vertices to the independent set. To our knowledge, METAMIS presents the state-of-the-art for Weighted Independent Set heuristics. Hence, local search with (restricted) $k$-swap neighborhoods is a crucial ingredient for currently leading heuristics for Weighted Independent Set.

**Our results.** We provide a complexity analysis for Weighted Independent Set with the $k$-swap neighborhood, denoted Weighted Independent Set/$k$-swap. Our main result is the PLS-completeness of Weighted Independent Set/3-swap on graphs of constant maximum degree.[1] We first show in Section 7.1 that Weighted Independent Set/7-swap is PLS-complete on graphs of maximum degree at most 6. Then, in Section 7.2, we extend the constructed instance of Weighted Independent Set/7-swap to obtain PLS-completeness even for Weighted Independent Set/3-swap on graphs of constant maximum degree.

Afterwards, we use the PLS-completeness for Weighted Independent Set/3-swap on graphs of constant maximum degree to show PLS-completeness for further problems: In Section 7.3, we consider Weighted $\Pi$ Subgraph/3-swap where the input is a graph $G$ and the aim is to find a maximum-weight vertex set $S$ such that the subgraph of $G$ induced by $S$ fulfills the property $\Pi$. We show PLS-completeness for Weighted $\Pi$ Subgraph/3-swap for any polynomial-time decidable graph property $\Pi$ that is closed under vertex deletion and under taking the disjoint union of graphs. In Section 7.4, we show PLS-completeness for Weighted Dominating Set/3-swap.

Finally, in Section 7.5, we show that if we allow all 3-swaps except either (i) the swaps that remove two vertices and add one or (ii) the swaps that remove one vertex and add two, we can find locally optimal solutions for Weighted Independent Set and Weighted Dominating Set in polynomial time. This implies that locally optimal solution for these problems can be found in polynomial time with respect to 2-swaps. Our algorithms are actually more general in two ways: First, they work for more general neighborhoods where either (i) all swaps that add up to $k$ vertices and remove at most one vertex are allowed or (ii) all swaps that remove up to $k$ vertices and add at most one vertex are allowed. Second, the algorithms can be applied to a more general type of subset-weight optimization problems that include the above-mentioned Weighted $\Pi$ Subgraph and Weighted Deletion to $\Pi$ problems.

## 7.1 Hardness of Finding 7-Optimal Independent Sets

To prove the PLS-completeness of Weighted Independent Set/3-swap on graphs of constant maximum degree, we first show PLS-completeness for Weighted Independent Set/7-swap and use the obtained graph as a starting point in a subsequent

[1] Our proof gives a degree bound of 3140.

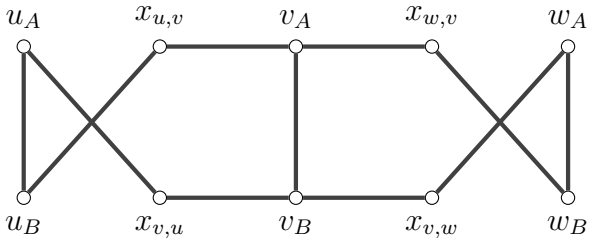



**Figure 7.1:** An example for the vertices and edges added to $G'$ for a vertex $v \in V$ with two neighbors $u$ and $w$ in $G$ in the reduction from Max Cut/flip to Weighted Independent Set/7-swap.

reduction to Weighted Independent Set/3-swap.

**Theorem 7.1.** Weighted Independent Set/7*-swap is* PLS*-complete on graphs of maximum degree 6.*

As mentioned above, Weighted Independent Set/$k$-swap is contained in PLS for each constant value of $k$. Hence, we only have to show that Weighted Independent Set/7-swap is PLS-hard.

**Construction.** We present a PLS-reduction from Max Cut/flip to Weighted Independent Set/7-swap. Let $I = (G = (V, E), \omega)$ be an instance of Max Cut/flip where $G$ has maximum degree 5. Even under these restrictions, Max Cut/flip is PLS-complete [47]. We describe how to construct an instance $I' = (G' = (V', E'), \omega')$ of Weighted Independent Set/7-swap in polynomial time and provide a polynomial-time computable solution-mapper $f$ from $I'$ to $I$ that preserves local optimality.

We start with an empty graph $G'$ and add, for each vertex $v \in V$, two new adjacent vertices $v_A$ and $v_B$ to $G'$. Next, for each edge $\{u, v\} \in E$, we add two new vertices $x_{u,v}$ and $x_{v,u}$ to $G'$ and make $x_{u,v}$ adjacent to $u_B$ and $v_A$ and $x_{v,u}$ adjacent to $u_A$ and $v_B$. This completes the construction of $G'$; Figure 7.1 shows an example. Note that $G'$ has a maximum degree of six. In the following, let $V_A := \{v_A \mid v \in V\}$ and $V_B := \{v_B \mid v \in V\}$. For each vertex $v \in V$, the vertices $v_A$ and $v_B$ will determine for each independent set $S$ of $G'$ whether vertex $v$ is contained in $A$ or $B$ for $(A, B) = f(S)$. Intuitively, since $G$ has maximum degree 5, a swap of vertex $v \in V$ in $G$ can then be simulated by a 7-swap in $G'$ that swaps $v_A$ and $v_B$ and all vertices corresponding to edges incident with vertex $v$ in $G$.

Next, we define the weight function $\omega' \colon V' \to \mathbb{N}$. Let $Z := \sum_{e \in E} \omega(e)$ denote the total weight of all edges. We set for each vertex $v \in V$, $\omega'(v_A) := \omega'(v_B) := 16 \cdot Z$

and for each edge $\{u, v\} \in E$, we set $\omega'(x_{u,v}) := \omega'(x_{v,u}) := 8 \cdot \omega(\{u, v\})$. In principle, the factors of 8 can be omitted but they will come in handy later when we present the PLS-reduction to Weighted Independent Set/3-swap.

This completes the construction of $I'$. It remains to define the solution-mapper $f$. For an independent set $S$, we define $f(S)$ to be the partition $(A, B)$ of $G$ where $A := \{v \in V \mid v_A \in S\}$ and $B := V \setminus A$. Recall that for vertex $v \in V$, the vertices $v_A$ and $v_B$ are adjacent in $G'$.

**Correctness.** To show the correctness of the reduction, we first analyze the structure of 7-optimal independent sets in $G'$. To this end, we define a notion of *nice* independent sets in $G'$ and show that all 7-optimal independent sets in $G'$ are nice. Afterwards, to show that the solution-mapper $f$ preserves local optimality, it thus remains to show that for each nice 7-optimal independent sets $S$, $f(S)$ is flip optimal. Informally, an independent set $S$ is nice, if it corresponds to a partition of $V$ based on the contained vertices of $\{v_A, v_B \mid v \in V\}$. Recall that if some vertex $v$ of $V$ is contained in $A$ for $f(S) = (A, B)$, then $v_B$ is not contained in $S$.

**Definition 7.2.** Let $S$ be an independent set in $G'$ and let $A := \{v \in V \mid v_A \in S\}$ and let $B := \{v \in V \mid v_B \in S\}$. We call $S$ *nice* if $(A, B)$ is a partition of $G$ and for each edge $\{u, v\} \in E$ with $(u, v) \in A \times B$, $S$ contains $x_{u,v}$.

**Lemma 7.3.** *If an independent set $S$ in $G$ is not nice, then $S$ is not 7-optimal in $G'$.*

*Proof.* Let $S$ be an independent set in $G'$ which is not nice and let $A := \{v \in V \mid v_A \in S\}$ and let $B := \{v \in V \mid v_B \in S\}$. We now show that $S$ is not 7-optimal.

First, assume $A \cup B \neq V$. Then, there is some vertex $v \in V$ with $v_A \notin S$ and $v_B \notin S$. Let $X_S^v := N_{G'}(v_A) \cap S$ denote the set of neighbors of $v_A$ in the current independent set $S$. Since $v_B \notin S$, the weight $\omega'(X_S^v)$ is at most $8 \cdot Z < \omega'(v_A)$. Hence, $W := X_S^v \cup \{v_A\}$ is an improving swap for $S$ in $G'$. Moreover, $W$ has size at most 6 because $G$ has maximum degree 5 which implies that $X_S^v$ has size at most 5. Second, assume $A \cup B = V$ and there is some edge $\{u, v\} \in E$ with $(u, v) \in A \times B$ where $x_{u,v} \notin S$. By construction, the vertex $x_{u,v}$ is adjacent to exactly the two vertices $u_B$ and $v_A$. Since $u_A \in S$ and $v_B \in S$, neither $u_B \in S$ nor $v_A \in S$. Hence, $\{x_{u,v}\}$ is an improving swap for $S$ in $G'$. □

Hence, in the following, we only have to consider nice independent sets in $G'$ when considering 7-optimal independent sets in $G'$. We are now able to show that the solution-mapper $f$ preserves local optimality.

**Lemma 7.4.** *Let $S$ be a nice independent set in $G'$. If $S$ is 7-optimal in $G'$, then $f(S)$ is flip-optimal in $G$.*

*Proof.* We show the statement by contraposition, that is, we show that $S$ is not 7-optimal in $G'$ if $f(S)$ is not flip-optimal in $G$. Let $(A, B) := f(S)$. By definition of $f$ and the fact that $S$ is nice, $A = \{v \in V \mid v_A \in S\}$ and $B = \{v \in V \mid v_B \in S\}$. Suppose that $(A, B)$ is not flip-optimal in $G$. Then, there is some vertex $v \in V$ where the total weight of the edges that are incident with $v$ and in the cut $E_G(A, B)$ is less than the total weight of the edges that are incident with $v$ and not in the cut $E_G(A, B)$. That is, either a) $v \in A$ and $\omega(E_G(\{v\}, A)) > \omega(E_G(\{v\}, B))$ or b) $v \in B$ and $\omega(E_G(\{v\}, B)) > \omega(E_G(\{v\}, A))$.

Without loss of generality we may assume that $v \in A$ and $\omega(E_G(\{v\}, A)) > \omega(E_G(\{v\}, B))$. Let $X_A := N_G(v) \cap A$ denote the neighbors of $v$ in $A$ and let $X_B := N_G(v) \cap B$ denote the neighbors of $v$ in $B$. Since $S$ is nice, we know that $x_{v,u} \in S$ for each $u \in X_B$. Moreover, for each $w \in X_A$, $x_{v,w} \notin S$ since $w_A \in S$ and $x_{w,v} \notin S$ since $v_A \in S$. We show that

$$W := \{v_A, v_B\} \cup \{x_{v,u} \mid u \in X_B\} \cup \{x_{w,v} \mid w \in X_A\}$$

is a valid improving 7-swap for $S$ in $G'$. An example of the swap $W$ is illustrated in Figure 7.2. First of all, note that $W$ has size at most 7 since $G$ has a maximum degree of 5 and thus $|X_A \cup X_B| \leq 5$. Moreover, by the fact that $\omega'(x_{y,z}) := \omega'(x_{z,y}) := 8 \cdot \omega(\{y, z\})$ for each edge $\{y, z\} \in E$,

$$\begin{aligned}\omega'(\{x_{w,v} \mid w \in X_A\}) &= 8 \cdot \omega(\{\{v, w\} \mid w \in X_A\}) \\ &> 8 \cdot \omega(\{\{v, u\} \mid u \in X_B\}) = \omega'(\{x_{v,u} \mid u \in X_B\}).\end{aligned}$$

Hence, $W$ is improving, since $\omega'(v_A) = \omega'(v_B)$. It remains to show that $W$ is valid. Since $W$ removes all neighbors of $v_B$ from $S$, $S \oplus W$ does not contain any neighbor of $v_B$. Recall that $v_A \notin S \oplus W$ and that for each vertex $w \in X_A$, $S \oplus W$ contains $w_A$ and avoids $w_B$. Hence, for each vertex $w \in X_A$, $x_{w,v}$ has no neighbor in $S \oplus W$. As a consequence, $W$ is valid and thus $S$ is not 7-optimal. □

Since Weighted Vertex Cover and Weighted Independent Set are dual problems, Observation 2.15 implies hardness also for Weighted Vertex Cover.

**Corollary 7.5.** Weighted Vertex Cover/7*-swap is* PLS*-complete on graphs of maximum degree 6.*

As a final remark, note that it is open whether Max Cut/flip is PLS-hard on graphs of maximum degree 4. If this is the case, the construction above would directly imply hardness for Weighted Independent Set/6-swap and Weighted Vertex Cover/6-swap graphs of maximum degree 5.

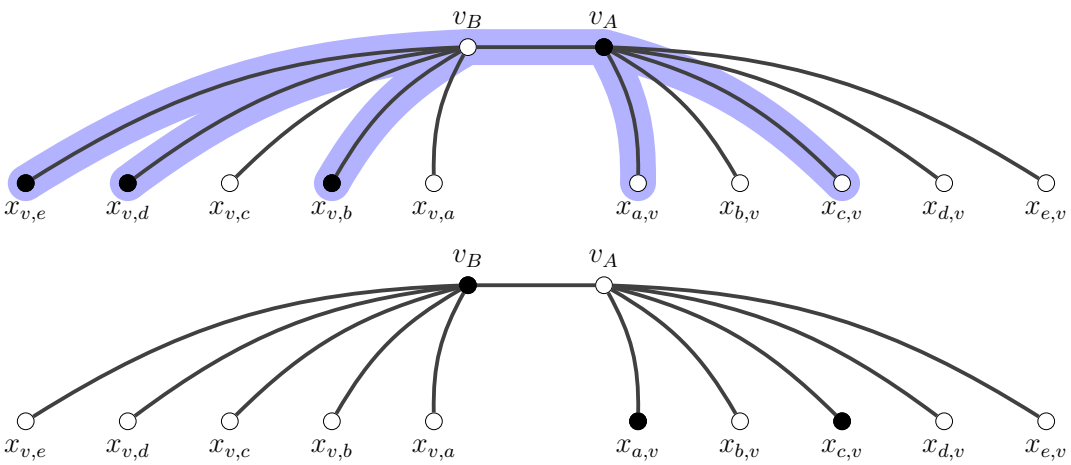



**Figure 7.2:** An example of an improving 7-swap $W$ for a nice independent set $S$ in $G'$ that simulates the flip of a vertex $v \in V$ from $A$ to $B$ in $G$, where $N_G(v) = \{a, b, c, d, e\}$ and $N_G(v) \cap B = \{b, d, e\}$. The black vertices are the vertices of $S$ and all vertices of $W$ are highlighted by the blue shape.

## 7.2 Hardness of Finding 3-Optimal Independent Sets

In this section, we use the previous reduction to also show that Weighted Independent Set/3-swap is PLS-complete. To this end, we introduce additional gadgets to the constructed graph.

**Theorem 7.6.** Weighted Independent Set/3-*swap is* PLS-*complete on graphs of constant maximum degree.*

**Construction.** To show this, we present a PLS-reduction from Max Cut/flip. As mentioned above, this reduction first computes the Weighted Independent Set/7-swap-instance described in the previous section. Afterwards, we add additional gadgets to this instance to ensure that we are able to simulate improving flips by a sequence of improving 3-swaps. We describe how to construct an instance $I'' = (G'' = (V'', E''), \omega'')$ of Weighted Independent Set/3-swap in polynomial time and provide a polynomial-time computable solution-mapper $f$ from $I''$ to $I$ that preserves local optimality. First, we compute the instance $I' = (G' = (V', E'), \omega')$ of Weighted Independent Set/7-swap described above. As above, let $V_A := \{v_A \mid v \in V\}$ and let $V_B := \{v_B \mid v \in V\}$. Moreover, we set for each

vertex $v \in V$, $X_v := \{x_{v,w}, x_{w,v} \mid w \in N(v)\}$, that is, $X_v$ is the set of vertices in $G'$ that were introduced for the incident edges of $v$ in $G$. We write $N_G(v)$ and $N_G[v]$ when considering the neighborhood of a vertex $v$ of $V$ in $G$ and $N(v)$ or $N[v]$ when considering the neighborhood of a vertex $v$ of $V''$ in $G''$.

Initially, we add edges such that for each vertex $v \in V$, $u \in N_G(v)$, and $w \in N_G(v)$, the vertices $x_{u,v}$ and $x_{v,w}$ are adjacent in $G''$. Note that this includes edges between $x_{u,v}$ and $x_{v,u}$ in $G''$ for each edge $\{u, v\} \in E$. Hence, in an independent set $S$ in $G''$, for each vertex $v \in V$, there is no vertex $x_{u,v} \in S$ if $S$ already contains a vertex $x_{v,w}$. The idea is that for each vertex $v$ of $G$, at most one of $v_A$ and $v_B$ has neighbors in $X_v \cap S$.

Next, we add gadgets to allow us to simulate flips in $G$ by a sequence of 3-swaps in $G''$. To this end, we first compute for each vertex $v \in V$ the collection $\mathcal{P}_v$ of subsets $P \subseteq N_G(v)$ fulfilling $\omega(E_G(\{v\}, P)) < \omega(E_G(\{v\}, N_G(v) \setminus P))$. Intuitively, in a partition $(A, B)$ of $G$, flipping a vertex $v$ from $A$ to $B$ is improving if and only if $N_G(v) \cap B$ is contained in $\mathcal{P}_v$. Note that $\emptyset \in \mathcal{P}_v$ and $N_G(v) \notin \mathcal{P}_v$ for each $v \in V$. Next, we add, for each nonempty $P \in \mathcal{P}_v$ a set of $|N_G(v)| - 1$ new vertices to $G''$. Let $Q := N_G(v) \setminus P$. Now, fix an ordering on both $P$ and $Q$ and let $P(i)$ denote the $i$th element of $P$ and let $Q(j)$ denote the $j$th element of $Q$ where $i \in [1, |P|]$ and $j \in [1, |Q|]$.

The newly added vertices are of three types: up, turn, and down. To simulate a flip of $v$ from $A$ to $B$, we have to remove the neighbors of $v_B$ from $S$, add $v_B$ to $S$, and add the vertices representing edges between $v$ and $A$. Intuitively, the up-vertices allow us—with a sequence of improving 3-swaps—to remove all neighbors of $v_B$ from $S$ except $v_A$ and the up-vertex of highest level. Afterwards, the turn-vertex allows us—with two improving 3-swaps—to remove $v_A$ from $S$ and add $v_B$ and the down-vertex of highest level to $S$. Finally, the down-vertices allow us—with a sequence of improving 3-swaps—to add the vertices to $S$ that represent the edges between $v$ and $A$. After all these 3-swaps, no intermediately added auxiliary vertex remains in $S$. In total, this allows us to simulate improving 7-swaps in $G'$ and thus improving flips in $G$ by a sequence of improving 3-swaps in $I''$. An example for a sequence of improving 3-swaps in $G''$ to simulate a flip in $G$ is illustrated in Figure 7.3.

First, we add for each $i \in [1, |P| - 1]$, a new vertex $\mathrm{up}_{v,P}^{A,i}$ to $G$ such that the neighborhood of $\mathrm{up}_{v,P}^{A,i}$ is exactly $X_v$ minus the vertices $\{x_{v,P(j)} \mid j < i\}$. Hence, $\mathrm{up}_{v,P}^{A,1}$ is adjacent to all vertices of $X_v$. Furthermore, we add an edge between $\mathrm{up}_{v,P}^{A,i}$ and each vertex of $\{w_A \mid w \in P\} \cup \{w_B \mid w \in Q\}$ and an edge between $\mathrm{up}_{v,P}^{A,i}$ and $v_B$.

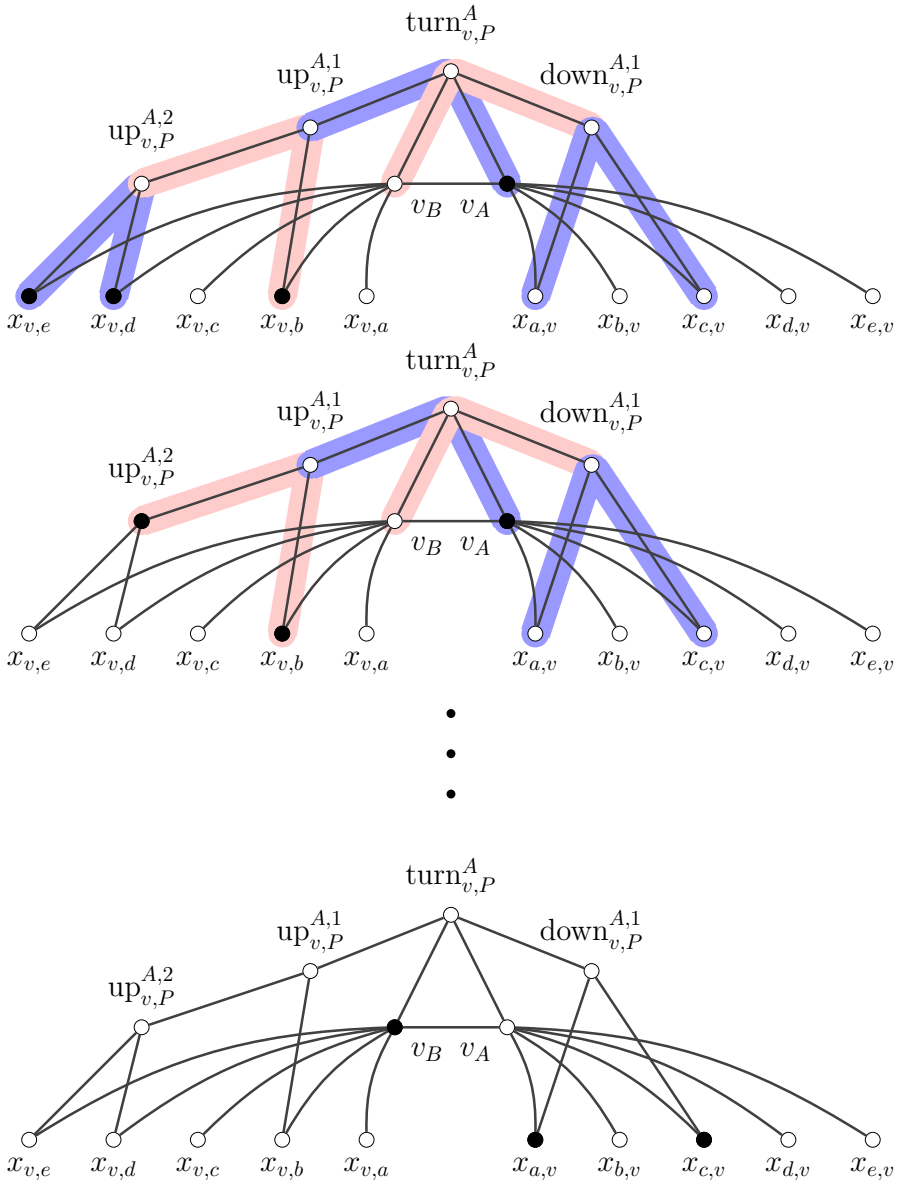



**Figure 7.3:** The simulation of the 7-swap shown in Figure 7.2 by a sequence of 3-swaps (from left to right highlighted alternating by either a blue or a red shape). Since the 7-swap shown in Figure 7.2 simulates a flip of a vertex $v \in V$ from $A$ to $B$ in $G$, where $N_G(v) = \{a, b, c, d, e\}$ and $P = N_G(v) \cap B = \{b, d, e\}$ with $P(1) = b$, $P(2) = d$, and $P(3) = e$, this sequence of 3-swaps also simulates the flip of vertex $v$ from $A$ to $B$. The black vertices are the vertices of the current independent set $S$. Note that not all edges of this subgraph are shown but only the important ones for the sequence of improving 3-swaps.

Moreover, we set

$$\omega''(\mathrm{up}_{v,P}^{A,i}) := 4 - i + 8 \cdot \sum_{j=i}^{|P|} \omega(\{v, P(j)\}) = 4 - i + \sum_{j=i}^{|P|} \omega''(x_{v,P(j)}).$$

These weights are defined in a way that for $i \geq 2$, removing the vertices $\mathrm{up}_{v,P}^{A,i}$ and $x_{v,P(j)}$ and adding vertex $\mathrm{up}_{v,P}^{A,i-1}$, increases the total weight of the solution by exactly 1. Intuitively, when flipping vertex $v$ from $A$ to $B$ in $G$ with $P := N_G(v) \cap B$, one can obtain a 3-neighbor for $S$ in $G$ as follows:

1. if $w_A \notin S$ for any neighbor $w \in N_G(v)$ in $P$ and $u_B \notin S$ for any neighbor $u \in N_G(v)$ in $Q$, then we remove the vertices $x_{v,P(|P|-1)}$ and $x_{v,P(|P|)}$ and add $\mathrm{up}_{v,P}^{A,|P|-1}$, and

2. if $\mathrm{up}_{v,P}^{A,j} \in S$ for some $j \in [2, |P| - 1]$, then we remove $\mathrm{up}_{v,P}^{A,j}$ and $x_{v,P(j)}$ and add $\mathrm{up}_{v,P}^{A,j-1}$.

Hence, during the simulation of an improving flip of a vertex $v$ in a partition $(A, B)$ from $A$ to $B$ in $G$, we can replace all vertices of $X_v$ by the vertex $\mathrm{up}_{v,P}^{A,1}$ with a sequence of 3-swaps. In the correctness proof of the reduction, we will show that all these 3-swaps are improving.

Recall that $Q = N_G(v) \setminus P$. Second, we add a vertex $\mathrm{turn}_{v,P}^{A}$ to $G''$ which is adjacent to all vertices of $X_v$, $\{v_A, v_B\}$, and to the vertices of $\{w_A \mid w \in P\} \cup \{w_B \mid w \in Q\}$. Recall that $\omega''(v_A) = \omega''(v_B) = 16 \cdot Z$ where $Z = \sum_{e \in E} \omega(e)$. We set

$$\omega''(\mathrm{turn}_{v,P}^{A}) := 4 + \omega''(v_A) + 8 \cdot \sum_{w \in P} \omega(\{v, w\}) = 4 + 16 \cdot Z + \sum_{w \in P} \omega''(x_{v,w}).$$

Intuitively, if $\mathrm{up}_{v,P}^{A,1}$ is contained in $S$, one can obtain an improving 3-neighbor for $S$ in $G$ by removing both $\mathrm{up}_{v,P}^{A,1}$ and $v_A$ from $S$ and adding $\mathrm{turn}_{v,P}^{A}$ to $S$.

Third, we add, similar to $\mathrm{up}_{v,P}^{A,i}$, for each $i \in [1, |Q| - 1]$ a new vertex $\mathrm{down}_{v,P}^{A,i}$ to $G$ such that the neighborhood of $\mathrm{down}_{v,P}^{A,i}$ is exactly $X_v$ minus the vertices $\{x_{Q(j),v} \mid j < i\}$. Hence, $\mathrm{down}_{v,P}^{A,1}$ is adjacent to all vertices of $X_v$. Furthermore, we add an edge between $\mathrm{down}_{v,P}^{A,i}$ and each vertex of $\{w_A \mid w \in P\} \cup \{w_B \mid w \in Q\}$ and we also add an edge between $\mathrm{down}_{v,P}^{A,i}$ and $v_A$. Moreover, we set

$$\omega''(\mathrm{down}_{v,P}^{A,i}) := i - 4 + 8 \cdot \sum_{j=i}^{|Q|} \omega(\{v, Q(j)\}) = i - 4 + \sum_{j=i}^{|Q|} \omega''(x_{v,Q(j)}).$$

Note that $\omega''(\text{down}^{A,i}_{v,P}) > 0$ since $Q \neq \emptyset$ and the images of $\omega$ are positive numbers. Intuitively, one can obtain an improving 3-neighbor for $S$ in $G$ as follows:

1. if $\text{turn}^{A}_{v,P} \in S$, then we remove $\text{turn}^{A}_{v,P}$ and add $\text{down}^{A,1}_{v,P}$ and $v_B$,

2. if $\text{down}^{A,j}_{v,P} \in S$ for some $j < |Q|-1$, then we remove $\text{down}^{A,j}_{v,P}$ and add $\text{down}^{A,j+1}_{v,P}$ and $x_{Q(j),v}$, and

3. if $\text{down}^{A,|Q|-1}_{v,P} \in S$, then we remove $\text{down}^{A,|Q|-1}_{v,P}$ and add $x_{Q(|Q|-1),v}$ and $x_{Q(|Q|),v}$.

With the current graph, it is possible to simulate a flip of a vertex $v$ from $A$ to $B$. To also simulate a flip of vertex $v$ from $B$ to $A$, we add symmetric vertices to $G''$, that is, for each vertex $\text{up}^{A,i}_{v,P}$, we add a vertex $\text{up}^{B,i}_{v,P}$, for each vertex $\text{turn}^{A}_{v,P}$, we add a vertex $\text{turn}^{B}_{v,P}$, and for each vertex $\text{down}^{A,i}_{v,P}$, we add a vertex $\text{down}^{B,i}_{v,P}$.

For the sake of completeness, we provide their formal definition. Let $P$ be a nonempty set of $\mathcal{P}_v$ and let $Q := N_v(G) \setminus P$.

- First, we add for each $i < |P|$, a new vertex $\text{up}^{B,i}_{v,P}$ to $G$ such that the neighborhood of $\text{up}^{B,i}_{v,P}$ is exactly $X_v$ minus the vertices $\{x_{P(j),v} \mid j < i\}$. Furthermore, we add an edge between $\text{up}^{B,i}_{v,P}$ and each vertex of $\{w_B \mid w \in P\} \cup \{w_A \mid w \in Q\}$ and we also add an edge between $\text{up}^{B,i}_{v,P}$ and $v_A$.

- Second, we add a vertex $\text{turn}^{B}_{v,P}$ to $G''$ which is adjacent to all vertices of $X_v$, $\{v_A, v_B\}$, and to the vertices of $\{w_B \mid w \in P\} \cup \{w_A \mid w \in Q\}$.

- Third, we add for each $i < |Q|$, a new vertex $\text{down}^{B,i}_{v,P}$ to $G$ such that the neighborhood of $\text{down}^{B,i}_{v,P}$ is exactly $X_v$ minus the vertices $\{x_{v,Q(j)} \mid j < i\}$. Furthermore, we add an edge between $\text{down}^{B,i}_{v,P}$ and each vertex of $\{w_B \mid w \in P\} \cup \{w_A \mid w \in Q\}$ and we also add an edge between $\text{down}^{B,i}_{v,P}$ and $v_B$.

- Moreover, we set $\omega''(\text{up}^{B,i}_{v,P}) := \omega''(\text{up}^{A,i}_{v,P})$ for each $i \in [1, |P|-1]$, $\omega''(\text{turn}^{B}_{v,P}) := \omega''(\text{turn}^{A}_{v,P})$, and $\omega''(\text{up}^{B,j}_{v,P}) := \omega''(\text{up}^{A,j}_{v,P})$ for each $j \in [1, |Q|-1]$.

Note that we did not add any vertices for $P = \emptyset \in \mathcal{P}_v$ to the graph $G''$. In the correctness proof, we show that a single improving 3-swap for $S$ is sufficient to simulate the flip of a vertex $v$ of $G$ if no edge incident with $v$ is currently in the cut.

For each vertex $v \in V$, let $V_v$ denote the set of the additional vertices associated with $v$, that is,

$$V_v := \bigcup_{P \in \mathcal{P}_v, P \neq \emptyset} (\{\mathrm{up}_{v,P}^{A,i}, \mathrm{up}_{v,P}^{B,i} \mid i < |P|\} \cup \{\mathrm{turn}_{v,P}^{A}, \mathrm{turn}_{v,P}^{B}\} \cup \{\mathrm{down}_{v,P}^{A,i}, \mathrm{down}_{v,P}^{B,i} \mid i < |N_G(v) \setminus P|\}).$$

To complete the construction, for each $v \in V$, we make the set $\bigcup_{w \in N_G[v]} V_w$ a clique in $G''$.

**Lemma 7.7.** *The graph $G''$ has maximum degree at most 3140.*

*Proof.* We show that $G''$ has a maximum degree of 3140. To this end, we first show that for each $v \in V$, the set $V_v$ contains at most 120 vertices. Since $G$ has maximum degree 5 and if some $P$ is contained in $\mathcal{P}_v$, then $N_G(v) \setminus P$ is not contained in $\mathcal{P}_v$, $\mathcal{P}_v$ contains at most $2^4$ subsets of $N_G(v)$. By construction, $V_v$ contains $|N_G(v)| - 1 \leq 4$ vertices for each nonempty $P \in \mathcal{P}_v$ and each $C \in \{A, B\}$. Hence, $V_v$ contains at most $(2^4 - 1) \cdot 2 \cdot 4 = 120$ vertices. Next, we show the maximum degree of $G''$. Let $v \in V$ and let $y$ be a vertex of $V_v \cup \{v_A, v_B\}$. Moreover, let $N_G^2[v] := \bigcup_{u \in N_G[v]} N_G[u]$ denote the set of vertices having distance at most two to $v$ in $G$. By definition, the neighborhood of $y$ in $G''$ is a subset of $X_v \cup \bigcup_{u \in N_G^2[v]} V_u \cup \bigcup_{u \in N_G(v)} \{u_A, u_B\}$. Since $G$ has maximum degree 5, $N_G^2[v]$ has size at most 26 and thus the degree of $y$ in $G''$ is at most $10 + 26 \cdot 120 + 10 = 3140$. Next, consider a vertex $x_{u,v}$ with $\{u, v\} \in E$. By construction, the neighborhood of $x_{u,v}$ is a subset of $\{u_B, v_A\} \cup V_u \cup V_v \cup X_v \cup X_u$. Hence, $x_{u,v}$ has degree at most $2 + 2 \cdot 120 + 20 = 262$. As a consequence, $G''$ has a maximum degree of 3140. □

It remains to define the solution-mapper $f$. Analogously to the solution-mapper of the presented PLS-reduction for Weighted Independent Set/7-swap, for an independent set $S$, we define $f(S)$ to be the partition $(A, B)$ of $G$ where $A := \{v \in V \mid v_A \in S\}$ and $B := V \setminus A$.

**Correctness.** To show the correctness of the PLS-reduction, we first analyze the structure of 3-optimal independent sets in $G''$. To this end, we show in Lemmas 7.8 – 7.10 that each 3-optimal independent set in $G''$ contains for each $v \in V$ either the vertex $v_A$ or the vertex $v_B$. Recall that the vertices $v_A$ and $v_B$ are adjacent in $G''$. As a consequence, this then implies that for $f(S) = (A, B)$, $B$ is exactly the set $\{v \in V \mid v_B \in S\}$. Finally, in Lemma 7.11, we then show that for such an independent set $S$, $f(S)$ is flip-optimal in $G$ if $S$ is 3-optimal in $I''$.

First, we show that no 3-optimal independent set in $G''$ contains any up-vertices.

**Lemma 7.8.** *Let $S$ be an independent set in $G''$. If $S$ contains a vertex $\mathrm{up}_{v,P}^{C,i}$ for $C \in \{A, B\}$, then $S$ is not 3-optimal.*

*Proof.* First, we show the statement for $i = 1$. By construction, the closed neighborhood $N[\mathrm{turn}_{v,P}^{C}]$ is exactly $N[\mathrm{up}_{v,P}^{C,1}] \cup \{v_C\}$ and $\omega''(\mathrm{turn}_{v,P}^{C}) = 1 + \omega''(\mathrm{up}_{v,P}^{C,1}) + \omega''(v_C)$. Hence, $S' := (S \cup \{\mathrm{turn}_{v,P}^{C}\}) \setminus \{\mathrm{up}_{v,P}^{C,1}, v_C\}$ is an improving 3-neighbor of $S$ in $G''$. This holds also if $v_C$ is not in $S$.

Second, we show the statement for $i > 1$. Let $r := x_{v,P(i-1)}$ if $C = A$ and $r := x_{P(i-1),v}$ if $C = B$. Note that by construction, the closed neighborhood $N[\mathrm{up}_{v,P}^{C,i-1}]$ is exactly $N[\mathrm{up}_{v,P}^{C,i}] \cup \{r\}$ and $\omega''(\mathrm{up}_{v,P}^{C,i-1}) = 1 + \omega''(\mathrm{up}_{v,P}^{C,i}) + \omega''(r)$. Hence, if an independent set $S$ contains $\mathrm{up}_{v,P}^{C,i}$ with $i > 1$, then $S$ is not 3-optimal in $G''$ since $(S \cup \{\mathrm{up}_{v,P}^{C,i-1}\}) \setminus \{\mathrm{up}_{v,P}^{C,i}, r\}$ is an improving 3-neighbor of $S$ in $G''$. This holds also if $r$ is not in $S$. □

Next, we show that each 3-optimal independent set in $G''$ contains for each vertex $v \in V$ a vertex of $\{v_A, v_B\}$ or a turn-vertex of $V_v$.

**Lemma 7.9.** *Let $S$ be an independent set in $G''$. If there is a vertex $v \in V$ such that $S$ avoids $v_A$, $v_B$, and $T_v := \{\mathrm{turn}_{v,P}^{A}, \mathrm{turn}_{v,P}^{B} \mid P \in \mathcal{P}_v, P \neq \emptyset\}$, then $S$ is not 3-optimal.*

*Proof.* Let $v \in V$ and let $S$ and $T_v$ be as specified in the lemma. Note that if for some $C \in \{A, B\}$, $S$ contains no vertex of $N(v_C)$, then $\{v_C\}$ is an improving 3-swap for $S$ in $G''$. Hence, we assume in the following that $N(v_A) \cap S \neq \emptyset$ and that $N(v_B) \cap S \neq \emptyset$. Recall that $N(v_A) \cup N(v_B) \subseteq X_v \cup \bigcup_{w \in N_G[v]} V_w$ and $\bigcup_{w \in N_G[v]} V_w$ is a clique in $G''$. Moreover, the common neighbors of $v_A$ and $v_B$ in $\bigcup_{w \in N_G[v]} V_w$ are the vertices of $T_v$. By assumption, $S$ contains no vertex of $T_v$. As a consequence, at most one vertex of $\{v_A, v_B\}$ has a neighbor in $(\bigcup_{w \in N_G[v]} V_w) \cap S$. Next, we analyze the neighbors of $v_A$ and $v_B$ in $X_v$. By construction, the neighborhood of $v_A$ in $X_v$ is $\{x_{w,v} \mid w \in N_G(v)\}$, and the neighborhood of $v_B$ in $X_v$ is $\{x_{v,w} \mid w \in N_G(v)\}$. These two sets are disjoint and each vertex of $N(v_A) \cap X_v$ is adjacent to each vertex of $N(v_B) \cap X_v$ by construction. As a consequence, at most one vertex of $v_A$ and $v_B$ has neighbors in $X_v \cap S$. Hence, one vertex of $v_A$ and $v_B$ has no neighbor in $X_v \cap S$ but has a neighbor in $(\bigcup_{w \in N_G[v]} V_w) \cap S$.

Assume without loss of generality that $v_A$ is the vertex of $\{v_A, v_B\}$ which has no neighbor in $X_v \cap S$ but has a neighbor $r$ in $(\bigcup_{w \in N_G[v]} V_w) \cap S$. In the following, we make a case distinction between the possible choices of $r$ and show in each case, that there is an improving 3-swap for $S$ in $G''$.

**Case:** $r = \text{up}_{w,i}^{C',P}$ for some $w \in N_G[v]$, some $C' \in \{A, B\}$, some nonempty set $P \in \mathcal{P}_w$, and some $i < |P|$**.** By Lemma 7.8, we directly obtain that $S$ is not 3-optimal in $G''$.

**Case:** $r = \text{turn}_{w,P}^{C'}$ for some $w \in N_G[v]$, some $C' \in \{A, B\}$, and some nonempty set $P \in \mathcal{P}_w$**.** Note that by assumption, $w \neq v$. Recall that $\omega''(\text{turn}_{w,P}^{C'}) < \omega''(v_A) + \omega''(w_A)$ and that $\text{turn}_{w,P}^{C'}$ is the only neighbor of $w_A$ in $S$, since $N[w_A] \subseteq N[\text{turn}_{w,P}^{C'}]$. Hence, since $\text{turn}_{w,P}^{C'}$ is the unique neighbor of $v_A$ in $S$, $(S \cup \{v_A, w_A\}) \setminus \{\text{turn}_{w,P}^{C'}\}$ is an improving 3-neighbor of $S$ in $G''$.

**Case:** $r = \text{down}_{w,i}^{C',P}$ for some vertex $w \in N_G[v]$, some $C' \in \{A, B\}$, some nonempty set $P \in \mathcal{P}_w$, and some $i < |N_w(G) \setminus P|$**.** Recall that $\omega''(\text{down}_{w,i}^{C',P}) < \omega''(v_A)$ and that $\text{down}_{w,i}^{C',P}$ is the only neighbor of $v_A$ in $S$. Hence, $(S \cup \{v_A\}) \setminus \{\text{down}_{w,i}^{C',P}\}$ is an improving 3-neighbor of $S$ in $G''$. □

Hence, we can assume that each 3-optimal independent set $S$ in $G''$ contains no up-vertex and for each vertex $v \in V$ exactly one vertex of $\{v_A, v_B\} \cup T_v$ with $T_v := \{\text{turn}_{v,P}^{A}, \text{turn}_{v,P}^{B} \mid P \in \mathcal{P}_v, P \neq \emptyset\}$. In the following, we call an independent set $S$ of $G''$ *nice* if $S \subseteq V' = V_A \cup V_B \cup \bigcup_{v \in V} X_v$ and if $S$ is a nice independent set for the instance $I'$ of Weighted Independent Set/7-swap. That is, if for $A := \{v \in V \mid v_A \in S\}$ and $B := \{v \in V \mid v_B \in S\}$, $(A, B)$ is a partition of $G$ and for each edge $\{u, v\} \in E$ with $(u, v) \in A \times B$, the vertex $x_{u,v}$ is contained in $S$. Next, we show similar to Lemma 7.4, that each 3-optimal independent set $S$ in $G''$ is nice.

**Lemma 7.10.** *Let $S$ be an independent set in $G''$. If $S$ is 3-optimal, then $S$ is nice.*

*Proof.* We show this statement by contraposition. That is, given an independent set $S$ in $G''$ which is not nice, we show that $S$ is not 3-optimal. Due to Lemma 7.8, we know that $S$ is not 3-optimal if some vertex $\text{up}_{v,P}^{C,i}$ for $C \in A, B$ is contained in $S$. Hence, in the following, we assume that none of these vertices is contained in $S$. Moreover, due to Lemma 7.9, we also know that $S$ is not 3-optimal if there is some vertex $v \in V$ such that $S$ does not contain $v_A$, $v_B$, and no vertex of $T_v = \{\text{turn}_{v,P}^{A}, \text{turn}_{v,P}^{B} \mid P \in \mathcal{P}_v, P \neq \emptyset\}$. Hence, in the following we further assume that for each vertex $v \in V$, $S$ contains one of these vertices.

We distinguish whether there is some vertex $v \in V$, such that $S$ contains a vertex of $T_v$. In both cases, we show that $S$ is not 3-optimal.

**Case 1:** There is some vertex $v \in V$, such that $S$ contains some vertex of $T_v$**.** Assume without loss of generality that $\text{turn}_{v,P}^{A} \in S$ and let $Q := N_G(v) \setminus P$. Recall that $\bigcup_{w \in N_G[v]} V_w$ is a clique in $G''$ which implies that for each vertex $w \in N_G(v)$, $S$ contains no vertex of $T_w \subseteq V_w$. Furthermore, since for each vertex $u \in V$, $S$ contains exactly one vertex of $\{u_A, u_B\} \cup T_u$, for each neighbor $w$ of $v$ in $G$, $S$

contains either $w_A$ or $w_B$. Moreover, $\text{turn}_{v,P}^A$ is adjacent to all vertices of $X_v$, all vertices of $\{w_A \mid w \in P\}$, and all vertices of $\{w_B \mid w \in Q\}$. As a consequence, for each vertex $w \in P$, $S$ contains $w_B$ and avoids $w_A$, and for each vertex $u \in Q$, $S$ contains $u_A$ and avoids $u_B$. Furthermore, $\text{turn}_{v,P}^A$ is the unique neighbor of $v_B$ in $S$. Recall that $P \neq N_G(v)$ which implies that $Q$ is nonempty. In the following, it is thus sufficient to distinguish between $|Q| = 1$ and $|Q| > 1$.

**Case 1.1: $|Q| = 1$.** Let $w$ denote the unique vertex of $Q$. We show that $S' := (S \cup \{v_B, x_{w,v}\}) \setminus \{\text{turn}_{v,P}^A\}$ is an improving 3-neighbor of $S$ in $G''$. By construction, $v_B$ and $x_{w,v}$ are nonadjacent in $G''$ and

$$\begin{aligned}\omega''(v_B) + \omega''(x_{w,v}) &= 16 \cdot Z + 8 \cdot \omega(\{v, w\}) = 16 \cdot Z + 8 \cdot \sum_{u \in Q} \omega(\{v, u\}) \\ &> 16 \cdot Z + 4 + 8 \cdot \sum_{u \in P} \omega(\{v, u\}) = \omega''(\text{turn}_{v,P}^A)\end{aligned}$$

which is true, since the images of $\omega$ are positive integers and $P \in \mathcal{P}_v$ which implies $\sum_{u \in Q} \omega(\{v, u\}) \geq \sum_{u \in P} \omega(\{v, u\}) + 1$. Hence, it remains to show that $S'$ is an independent set. Since by definition, $N[v_B] \subseteq N[\text{turn}_{v,P}^A]$, we only have to show that $\text{turn}_{v,P}^A$ is the only neighbor of $x_{w,v}$ in $S$. By construction, the neighborhood of $x_{w,v}$ in $G''$ is a subset of $\{v_A, w_B\} \cup V_v \cup V_w \cup X_v \cup X_w$. Recall that $\text{turn}_{v,P}^A$ is adjacent to all vertices of this set except for some vertices of $X_w$. Hence, we only have to consider the neighbors of $x_{w,v}$ in $X_w$. By construction, these are the vertices $x_{u,w}$ where $u \in N_G(w)$. Since $w_A$ is contained in $S$ and each of these vertices $x_{u,w}$ is adjacent to $w_A$ in $G''$, $x_{w,v}$ has no neighbor in $S$ besides $\text{turn}_{v,P}^A$. As a consequence, $S'$ is an improving 3-neighbor of $S$.

**Case 1.2: $|Q| > 1$.** Then, the vertex $\text{down}_{v,P}^{A,1}$ exists. We show that $S' := (S \cup \{v_B, \text{down}_{v,P}^{A,1}\}) \setminus \{\text{turn}_{v,P}^A\}$ is an improving 3-neighbor of $S$ in $G''$. By construction, $v_B$ and $\text{down}_{v,P}^{A,1}$ are nonadjacent in $G''$ and

$$\begin{aligned}\omega''(v_B) + \omega''(\text{down}_{v,P}^{A,1}) &= 16 \cdot Z - 3 + 8 \cdot \sum_{w \in Q} \omega(\{v, w\}) \\ &> 16 \cdot Z + 4 + 8 \cdot \sum_{u \in P} \omega(\{v, u\}) = \omega''(\text{turn}_{v,P}^A)\end{aligned}$$

since the images of $\omega$ are positive integers and $P \in \mathcal{P}_v$ which yields $\sum_{u \in Q} \omega(\{v, u\}) > \sum_{u \in P} \omega(\{v, u\})$. Hence, it remains to show that $S'$ is an independent set. By definition, $N[v_B] \subseteq N[\text{turn}_{v,P}^A]$ and $N[\text{down}_{v,P}^{A,1}] \subseteq N[\text{turn}_{v,P}^A]$, which implies that $\text{turn}_{v,P}^A$ is the unique neighbor of both $v_B$ and $\text{turn}_{v,P}^A$ in $S$. As a consequence, $S'$ is an independent set and thus an improving 3-neighbor of $S$.

**Case 2:** For each vertex $v \in V$, $S$ contains either $v_A$ or $v_B$**.** In a first step, we show that if for some vertex $v \in V$, there is some vertex $r \in V_v$ contained in $S$, then $S$ is not 3-optimal. Note that we already showed that this is the case if $r$ is an up-vertex or a turn-vertex. Hence, it remains to show the claim for $r$ being a down-vertex. Assume without loss of generality that there is a nonempty set $P \in \mathcal{P}_v$ and some $i < |Q|$ with $Q := N_G(v) \setminus P$ such that $\text{down}_{v,P}^{A,i}$ is contained in $S$.

We distinguish between the cases of $i < |Q| - 1$ and $i = |Q| - 1$.

**Case 2.1:** $i < |Q| - 1$**.** Let $w$ denote the vertex $Q(i)$. We show that $S' := (S \cup \{\text{down}_{v,P}^{A,i+1}, x_{w,v}\}) \setminus \{\text{down}_{v,P}^{A,i}\}$ is an improving 3-neighbor of $S$ in $G''$. By construction, $\text{down}_{v,P}^{A,i+1}$ and $x_{w,v}$ are nonadjacent in $G''$ and

$$\begin{aligned}\omega''(\text{down}_{v,P}^{A,i+1}) + \omega''(x_{w,v}) &= i + 1 - 4 + \sum_{j=i+1}^{|Q|} \omega''(x_{Q(j),v}) + \omega''(x_{w,v}) \\ &> i - 4 + \sum_{j=i}^{|Q|} \omega''(x_{Q(j),v}) = \omega''(\text{down}_{v,i}^{A,P}).\end{aligned}$$

Hence, it remains to show that $S'$ is an independent set in $G''$. Note that by construction, $N[\text{down}_{v,P}^{A,i+1}] = N[\text{down}_{v,P}^{A,i}] \setminus \{x_{w,v}\}$. Consequently, we only have to show that $\text{down}_{v,P}^{A,i}$ is the only neighbor of $x_{w,v}$ in $S$. By construction, $N[x_{w,v}] \subseteq \{v_A, w_B\} \cup V_v \cup V_w \cup X_v \cup X_w$. Since $V_v \cup V_w$ is a clique in $G''$ and $\text{down}_{v,P}^{A,i}$ is a vertex of $V_v$ and $\text{down}_{v,P}^{A,i}$ is adjacent to $v_A$ and $w_B$, we only have to consider the neighbors of $x_{w,v}$ in $X_v \cup X_w$. By construction, the neighbors of $x_{w,v}$ in $X_v$ are the vertices $x_{v,u}$ where $u \in N_G(v)$ which are all adjacent to $\text{down}_{v,P}^{A,i}$. Hence, $S$ contains no neighbor of $x_{w,v}$ in $X_v$. Moreover, the neighbors of $x_{w,v}$ in $X_w$ are the vertices $x_{u,w}$ where $u \in N_G(w)$. For each such $u \in N_G(w)$, the vertex $x_{u,w}$ is adjacent to $w_A$ in $G''$. Recall that $S$ contains either $w_A$ or $w_B$. Since $w_B$ is adjacent to $\text{down}_{v,P}^{A,i}$, $S$ contains $w_A$, which implies that no vertex $x_{u,w}$ with $u \in N_G(v)$ is contained in $S$. As a consequence, both $x_{w,v}$ and $\text{down}_{v,P}^{A,i}$ have no neighbor in $S$ besides $\text{down}_{v,P}^{A,i}$. Hence, $S'$ is an improving 3-neighbor of $S$.

**Case 2.2:** $i = |Q| - 1$**.** Let $w_1$ denote the vertex $Q(i)$ and let $w_2$ denote the vertex $Q(i+1) = Q(|Q|)$. We show that $S' := (S \cup \{x_{w_1,v}, x_{w_2,v}\}) \setminus \{\text{down}_{v,P}^{A,|Q|-1}\}$ is an improving 3-neighbor of $S$ in $G''$. By construction, $x_{w_1,v}$ and $x_{w_2,v}$ are nonadjacent in $G''$ and

$$\omega''(x_{w_1,v}) + \omega''(x_{w_2,v}) + |Q| - 1 - 4 = \omega''(\text{down}_{v,|Q|-1}^{A,P}).$$

Since $P \neq \emptyset$ and $G$ has maximum degree 5, $Q$ has size at most 4, which implies that $S'$ is improving. Hence, it remains to show that $S'$ is an independent set in $G''$.

To this end, we show that for each $w \in \{w_1, w_2\}$, $\text{down}_{v,P}^{A,|Q|-1}$ is the only neighbor of $x_{w,v}$ in $S$. Let $w \in \{w_1, w_2\}$. By construction, the neighborhood of $x_{w,v}$ in $G''$ is a subset of $\{v_A, w_B\} \cup V_v \cup V_w \cup X_v \cup X_w$. Since $\text{down}_{v,P}^{A,|Q|-1}$ is in $S$, a vertex of the clique $V_v \cup V_w$ in $G''$, and adjacent to $v_A$ and $w_B$, it remains to show that $w$ has no neighbor in $(X_v \cup X_w) \cap S$. By construction, the neighbors of $x_{w,v}$ in $X_v$ are the vertices $x_{v,u}$ where $u \in N_G(v)$ which are all adjacent to $\text{down}_{v,P}^{A,|Q|-1}$. Hence, $S$ contains no neighbor of $x_{w,v}$ in $X_v$. Moreover, the neighbors of $x_{w,v}$ in $X_w$ are the vertices $x_{u,w}$ where $u \in N_G(w)$. For each such $u \in N_G(w)$, the vertex $x_{u,w}$ is adjacent to $w_A$ in $G''$. Recall that $S$ contains either $w_A$ or $w_B$. Since $w_B$ is adjacent to $\text{down}_{v,P}^{A,|Q|-1}$ and $\text{down}_{v,P}^{A,|Q|-1}$ is contained in $S$, $w_A$ is contained in $S$. This implies that no vertex $x_{u,w}$ with $u \in N_G(v)$ is contained in $S$. As a consequence, $x_{w_1,v}$ and $x_{w_2,v}$ have no neighbor in $S$ besides $\text{down}_{v,P}^{A,|Q|-1}$. Hence, $S'$ is an improving 3-neighbor of $S$.

Summarizing, we can assume in the following that $S \subseteq V' = V_A \cup V_B \cup \bigcup_{v \in V} X_v$ and that $(A, B)$ is a partition of $G$, where $A := \{v \in V \mid v_A \in S\}$ and $B := \{v \in V \mid v_B \in S\}$. Hence, it remains to show that if there is some edge $\{v, w\} \in E$ with $(v, w) \in A \times B$ such that $x_{v,w}$ is not contained in $S$, then $S$ is not 3-optimal in $G''$. To this end, we show that $S \cup \{x_{v,w}\}$ is an independent set in $G''$. By construction, $x_{v,w}$ is adjacent to $v_B$, $w_A$, some vertices of $V_v \cup V_w$, and the vertices $\{x_{u,v} \mid u \in N_G(v)\} \cup \{x_{w,u} \mid u \in N_G(w)\}$. Recall that $S$ contains no vertex of $V_v \cup V_w$. Moreover, since $(v, w) \in A \times B$, $v_A$ and $w_B$ are contained in $S$ which implies that $v_B$ and $w_A$ are not contained in $S$. Furthermore, since all vertices of $\{x_{u,v} \mid u \in N_G(v)\}$ are adjacent to $v_A$, all vertices of $\{x_{w,u} \mid u \in N_G(w)\}$ are adjacent to $w_B$, and $v_A$ and $w_B$ are contained in $S$, $S \cup \{x_{v,w}\}$ is an independent set in $G''$.

We conclude that if $S$ is not nice in $G''$, then $S$ is not 3-optimal. □

With the knowledge that each 3-optimal independent set in $G''$ is nice, we are now able to show the correctness of the PLS-reduction, that is, to show that $f$ preserves local optimality.

**Lemma 7.11.** *Let $S$ be a nice independent set in $G''$. If $S$ is 3-optimal in $G''$, then $f(S)$ is flip-optimal in $G$.*

*Proof.* We show the statement by contraposition. That is, we show that if $(A, B)$ is not flip-optimal in $G$, then $S$ is not 3-optimal in $G''$. Let $S$ be a nice independent set in $G''$ and let $A := \{v \in V \mid v_A \in S\}$ and $B := \{v \in V \mid v_B \in S\}$, such that $f(S) = (A, B)$ is not flip-optimal in $G$. Then, there is some vertex $v \in V$ where the total weight of the edges in the cut of $(A, B)$ incident with $v$ is less than the total weight of the edges incident with $v$ that are not in the cut of $(A, B)$. That is, either

a) $v \in A$ and $\omega(E_G(\{v\}, A)) > \omega(E_G(\{v\}, B))$ or b) $v \in B$ and $\omega(E_G(\{v\}, B)) > \omega(E_G(\{v\}, A))$.

Without loss of generality we may assume that $v \in A$ and $\omega(E_G(\{v\}, A)) > \omega(E_G(\{v\}, B))$. Hence, for $P := N_G(v) \cap B$ and $Q := N_G(v) \cap A$, $\sum_{u \in P} \omega(\{v, u\}) < \sum_{w \in Q} \omega(\{v, w\})$ which implies that $P \in \mathcal{P}_v$. Note that this further implies that $Q$ is nonempty. In the following, we distinguish between the cases of $|P| = 0$, $|P| = 1$, and $|P| \geq 2$ and present for each of these cases an improving 3-neighbor for $S$ in $G''$.

**Case:** $|P| = 0$**.** Let $w$ be an arbitrary neighbor of $v$ in $G$. Such a vertex exists, since $Q$ is nonempty. Since $w \in Q$, $w_A$ is contained in $S$. We show that $S' := (S \cup \{v_B, x_{w,v}\}) \setminus \{v_A\}$ is an improving 3-neighbor of $S$ in $G''$. Since $\omega''(v_A) = \omega''(v_B)$ and $\omega''(x_{w,v}) > 0$, it remains to show that $S'$ is an independent set in $G''$. Note that $v_B$ and $x_{w,v}$ are nonadjacent in $G''$ and that $S$ contains no vertex of $V_v \cup X_v$ since $S$ is nice and $P = \emptyset$. Hence, $v_A$ is the only neighbor of $v_B$ in $S$. Thus, we only have to show that $v_A$ is also the only neighbor of $x_{w,v}$ in $S$. By construction, $x_{w,v}$ is adjacent to $v_A$, $w_B$, some vertices of $V_v \cup V_w$ and the vertices $\{x_{v,u} \mid u \in N_G(v)\} \cup \{x_{u,w} \mid u \in N_G(w)\}$. Since $S$ is nice and contains $w_A$, $S$ contains no vertex of $V_v \cup V_w$ and no vertex of $X_v$. Hence, we only have to consider the vertices $x_{u,w}$ where $u \in N_G(w)$. Since each such vertex is adjacent to $w_A$, $v_A$ is the unique neighbor of $x_{w,v}$ in $S$. As a consequence, $S'$ is an improving 3-neighbor of $S$ in $G''$.

**Case:** $|P| = 1$**.** Let $w$ be the unique neighbor of $v$ in $B$, that is, the unique vertex of $P$. Note that $x_{v,w}$ is the only vertex of $X_v$ in $S$. Since $P$ is nonempty, there is a vertex $\text{turn}^A_{v,P}$ in $G''$ which is not contained in $S$ since $S$ is nice. We show that $S' := (S \cup \{\text{turn}^A_{v,P}\}) \setminus \{v_A, x_{v,w}\}$ is an improving 3-neighbor of $S$ in $G''$. Note that

$$\omega''(v_A) + \omega''(x_{v,w}) = 16 \cdot Z + \sum_{u \in P} \omega''(x_{v,w}) = \omega''(\text{turn}^A_{v,P}) - 4.$$

Hence, we only have to show that $S'$ is an independent set in $G''$. For this it is sufficient to show that $v_A$ and $x_{v,w}$ are the only two neighbors of $\text{turn}^A_{v,P}$ in $S$. Recall that $\text{turn}^A_{v,P}$ is adjacent to $v_A$, $v_B$, all vertices of $X_v$, the vertices $\{u_A \mid u \in P\}$, the vertices $\{u_B \mid u \in Q\}$ and some vertices of $\bigcup_{u \in V} V_u$. Since $v_A$ is contained in $S$, $v_B$ is not contained in $S$. Moreover, recall that $x_{v,w}$ is the only vertex of $X_v$ in $S$. Furthermore, since $S$ is nice, $S$ contains no vertex of $\bigcup_{u \in V} V_u$. Hence, it remains to show that $S$ contains no vertex of $\{u_A \mid u \in P\} \cup \{u_B \mid u \in Q\}$. By definition of $(A, B)$ and the fact that $P \subseteq B$ and $Q \subseteq A$, for each vertex $u \in P$, $u_B$ is contained in $S$ and for each vertex $u \in Q$, $u_A$ is contained in $S$. Hence, $S$ contains none of the vertices of $\{u_A \mid u \in P\} \cup \{u_B \mid u \in Q\}$ which implies that $S'$ is an improving 3-neighbor of $S$ in $G''$.

**Case:** $|P| \geq 2$**.** Let $i := |P| - 1$, let $w_1$ denote the vertex $P(i)$, and let $w_2$ denote the vertex $P(i+1)$. Note that $x_{v,w_1}$ and $x_{v,w_2}$ are contained in $S$ since $S$ is nice, $v \in A$, and $w_1$ and $w_2$ are in $P \subseteq B$. Moreover, since $i > 0$, there is a vertex $\mathrm{up}_{v,i}^{A,P}$ in $G''$ which is not contained in $S$. We show that $S' := (S \cup \{\mathrm{up}_{v,i}^{A,P}\}) \setminus \{x_{v,w_1}, x_{v,w_2}\}$ is an improving 3-neighbor of $S$ in $G''$. Note that

$$\begin{aligned}\omega''(x_{v,w_1}) + \omega''(x_{v,w_2}) &= \sum_{j=i}^{|P|} \omega''(x_{v,P(j)}) \\ &< \sum_{j=i}^{|P|} \omega''(x_{v,P(j)}) + 4 - i = \omega''(\mathrm{up}_{v,i}^{A,P})\end{aligned}$$

since $i < 4$ by the fact that $P \subseteq N_G(v)$ and $G$ has maximum degree 5. Hence, we only have to show that $S'$ is an independent set in $G''$. To this end, we show that $x_{v,w_1}$ and $x_{v,w_2}$ are the only two neighbors of $\mathrm{up}_{v,i}^{A,P}$ in $S$. Recall that $\mathrm{up}_{v,i}^{A,P}$ is adjacent to $v_B$, all vertices of $X_v$ except $x_{v,P(j)}$ for $j < i$, the vertices $\{u_A \mid u \in P\}$, the vertices $\{u_B \mid u \in Q\}$ and some vertices of $\bigcup_{u \in V} V_u$. Since $v_A$ is contained in $S$, we have that $v_B$ is not contained in $S$ and for each $u \in N_G(v)$, $x_{u,v}$ is not contained in $S$. By the fact that for each $u \in Q$, $u_A$ is contained in $S$, the vertex $x_{v,u}$ is not contained in $S$. Hence, $x_{v,w_1}$ and $x_{v,w_2}$ are the only two neighbors of $\mathrm{up}_{v,i}^{A,P}$ from $X_v$ in $S$. Moreover, since $S$ is nice, $S$ contains no vertex of $\bigcup_{u \in V} V_u$. Hence, it remains to show that $S$ contains no vertex of $\{u_A \mid u \in P\} \cup \{u_B \mid u \in Q\}$. By definition of $(A, B)$ and the fact that $P \subseteq B$ and $Q \subseteq A$, for each $u \in P$, $u_B$ is contained in $S$ and for each $u \in Q$, $u_A$ is contained in $S$. Hence, $S$ contains none of the vertices of $\{u_A \mid u \in P\} \cup \{u_B \mid u \in Q\}$ which implies that $S'$ is an improving 3-neighbor of $S$ in $G''$.

Concluding, for each nice 3-optimal independent set $S$ in $G''$, $f(S)$ is flip-optimal in $G$. □

Recall that to prove Theorem 7.6, we have to show that for each 3-optimal independent set $S$ in $G''$, $f(S)$ is flip-optimal in $G$. We obtain this by the following facts: a) due to Lemma 7.10, each 3-optimal independent set in $G''$ is nice and b) due to Lemma 7.11, for each nice 3-optimal independent set $S$ in $G''$, $f(S)$ is flip-optimal in $G$.

Again, since Weighted Vertex Cover and Weighted Independent Set are dual problems, Observation 2.15 implies hardness also for Weighted Vertex Cover.

**Corollary 7.12.** Weighted Vertex Cover*/3-swap is* PLS*-complete on graphs of constant maximum degree.*

## 7.3 Hardness of Finding 3-Optimal Solutions for Weighted Subgraph Deletion Problems

We now consider more general graph-based subset-weight optimization problems and extend the PLS-hardness with respect to the 3-swaps-neighborhood to these problems.

A *graph property* $\Pi$ is a collection of undirected and finite graphs. If a graph $G$ is contained in $\Pi$, we say that $G$ *fulfills (the graph property)* $\Pi$. In this section, we consider the following two general problems which are dual to each other.

Weighted $\Pi$ Subgraph
**Input:** A graph $G = (V, E)$ and a vertex-weight function $\omega\colon V \to \mathbb{N}$.
**Output:** A vertex set $S$ of maximum total weight such that $G[S]$ fulfills $\Pi$.

Weighted Deletion to $\Pi$
**Input:** A graph $G = (V, E)$ and a vertex-weight function $\omega\colon V \to \mathbb{N}$.
**Output:** A vertex set $S$ of minimum total weight such that $G - S$ fulfills $\Pi$.

We show that both Weighted $\Pi$ Subgraph/3-swap and Weighted Deletion to $\Pi$/3-swap are PLS-hard for each nontrivial hereditary graph property $\Pi$ where each minimal forbidden induced subgraph is connected. To this end, we first define these restricted graph properties.

We call a graph property $\Pi$ *nontrivial* if infinitely many graphs fulfill $\Pi$ and infinitely many graphs do not fulfill $\Pi$. We say that $\Pi$ is *hereditary (on induced subgraphs)* if for each graph $G$ that fulfills $\Pi$, each induced subgraph of $G$ also fulfills $\Pi$. We say that a graph $G$ is a *minimal forbidden induced subgraph* of a graph property $\Pi$ if $G$ does not fulfill $\Pi$ and each proper induced subgraph of $G$ fulfills $\Pi$. A graph property $\Pi$ is hereditary if and only if $\Pi$ can be characterized by a set $\mathcal{F}$ of minimal forbidden induced subgraphs. That is, a graph $G$ fulfills $\Pi$ if and only if $G$ does not contain any graph of $\mathcal{F}$ as an induced subgraph.

For example, the collection of all edgeless graphs is a hereditary graph property. This graph property has a single minimal forbidden induces subgraph: the graph that consists of a single edge and its both endpoints. For this graph property $\Pi$, Weighted Deletion to $\Pi$ is exactly Weighted Vertex Cover. Another hereditary graph property is the collection of all acyclic graphs. For this graph property the set of minimal forbidden induces subgraphs are exactly the collection of all cycle graphs.

In the remaining section, we establish the PLS-hardness for Weighted Π Subgraph/3-swap and Weighted Deletion to Π/3-swap.

**Theorem 7.13.** *Let* Π *be a nontrivial hereditary graph property where each minimal forbidden induced subgraph is connected. Then,* Weighted Π Subgraph/3*-swap and* Weighted Deletion to Π/3*-swap are* PLS*-hard even on graphs of constant maximum degree.*

Recall that due to Corollary 7.12 Weighted Vertex Cover/3-swap is PLS-complete on graphs of constant maximum degree. We present a PLS-reduction from Weighted Vertex Cover/3-swap to Weighted Deletion to Π/3-swap. Since Weighted Π Subgraph and Weighted Deletion to Π are dual subset-weight optimization problems, due to Observation 2.15, the PLS-hardness then directly transfers also to Weighted Π Subgraph.[2] In our reduction, we follow the general idea of the NP-hardness reduction from Vertex Cover to Deletion to Π presented by Lewis and Yannakakis [115].

**Definition of the $\alpha$-sequence.** Similar to Lewis and Yannakakis [115], we also define the notion of $\alpha$-sequences. Informally, an $\alpha$-sequence of a graph $G$ is a nonincreasingly ordered sequence of integers indicating how large the connected components of $G-v$ are for some suitably chosen vertex $v$ of $G$. Lewis and Yannakakis [115] used $\alpha$-sequences only for connected graphs. Based on our more restricted graph property Π, we also define $\alpha$-sequences for disconnected graphs. The rough idea for the reduction is to replace the edges in the Weighted Vertex Cover-instance by a graph $H$ that has lexicographically smallest $\alpha$-sequence among all graphs not fulfilling Π. In the following, let $\prec$ denote the lexicographic order relation. Moreover, we write $x \preceq y$ if $x = y$ or $x \prec y$.

For a graph $G = (V, E)$ with $\ell$ connected components, we let $\mathrm{comp}(G) = (|V_1|, \ldots, |V_\ell|)$ with $|V_i| \geq |V_{i+1}|$ for each $i \in [1, \ell - 1]$ denote the (nonincreasingly sorted) component-size vector of $G$. Now, assume that $G = (V, E)$ has at least two vertices. We denote by $\alpha(G)$ the *$\alpha$-sequence of $G$*, which is defined by $\alpha(G) := \mathrm{comp}(G - v)$, where $v$ is a vertex of $V$ such that $\mathrm{comp}(G - v) \preceq \mathrm{comp}(G - w)$ for each vertex $w \in V$. Observe that for every graph $G = (V, E)$ with at least two vertices and every vertex $v \in V$, we have $\mathrm{comp}(G-v) \prec \mathrm{comp}(G)$ since the deletion

[2]Note that if Π is not polynomial-time checkable, Weighted Deletion to Π/3-swap and Weighted Π Subgraph/3-swap are no subset-weight optimization problems. Still, the remaining properties of dual optimizations problems hold. Hence, Observation 2.15 still applies if Π is not polynomial-time checkable.

of $v$ decreases the size of its connected component by one or splits this component into several smaller components.

**Lemma 7.14.** *Let $G$ be a graph. Then, for each proper induced subgraph $\widehat{G}$ of $G$ with at least two vertices, $\alpha(\widehat{G}) \prec \alpha(G)$.*

*Proof.* We show the statement by induction over the number of vertices of $G = (V, E)$. For the base case where $G$ contains at most two vertices, the statement follows trivially, since there is no proper subgraph of $G$ with at least two vertices.

For the inductive step assume that $G$ has at least three vertices and the statement holds for each graph with fewer vertices than $G$. Let $v$ be a vertex of $V$ such that $\alpha(G) = \text{comp}(G - v)$. Moreover, let $\widehat{G} = (\widehat{V}, \widehat{E})$ be a proper and induced subgraph of $G$ with exactly one vertex less than $G$. We show that $\alpha(\widehat{G}) \prec \alpha(G)$. If $\widehat{G}$ contains the vertex $v$, let $w$ be the unique vertex of $V \setminus \widehat{V}$. Otherwise, that is, if $v$ is the unique vertex of $V \setminus \widehat{V}$, let $w$ be an arbitrary vertex of $\widehat{V}$. In both cases, $\widehat{V}$ contains exactly one of $v$ and $w$. Consider the graph $G - \{v, w\}$ and observe that $G - \{v, w\} = (G - w) - v = (G - v) - w$. Now the claim follows from two observations: First, $\text{comp}(G - \{v, w\}) \prec \text{comp}(G - v) = \alpha(G)$ since any vertex deletion results in a smaller component-size vector as observed above. Second, $\alpha(\widehat{G}) \preceq \text{comp}(G - \{v, w\})$ since the graph $G - \{v, w\}$ can be obtained from $\widehat{G}$ by deletion of the unique vertex in $\widehat{V} \cap \{v, w\}$. Together, this implies $\alpha(\widehat{G}) \preceq \text{comp}(G - \{v, w\}) \prec \alpha(G)$.

Hence, for each proper induced subgraph $\widehat{G}$ of $G$ with exactly one vertex less than $G$, $\alpha(\widehat{G}) \prec \alpha(G)$. Note that each proper induced subgraph of $G$ either has exactly one vertex less than $G$ or is a proper subgraph of some proper induced subgraph $\widehat{G}$ of $G$ with exactly one vertex less than $G$. Hence, the induction hypothesis implies that for each proper induced subgraph $\widehat{G}$ of $G$ with at least two vertices, $\alpha(\widehat{G}) \prec \alpha(G)$. □

**Definition of the gadget graphs $H$, $J$ and $D$.** Note that since $\Pi$ is nontrivial and hereditary on forbidden induced subgraphs and Weighted Deletion to $\Pi$ is a minimization problem, a graph consisting of a single vertex fulfills $\Pi$. Hence, each graph not fulfilling property $\Pi$ contains at least two vertices. Further, since $\Pi$ is nontrivial, there is a graph that does not fulfill $\Pi$.

Let $H$ be a minimal forbidden induced subgraph of $\Pi$ such that no other minimal forbidden induced subgraph of $\Pi$ has a lexicographically smaller $\alpha$-sequence. Furthermore, Lemma 7.14 implies that for each graph $H'$ with $\alpha(H') \prec \alpha(H)$, $H'$ fulfills $\Pi$. Note that $H$ is connected.

Let $c$ be a vertex of $H$ such that $\alpha(H) = \text{comp}(H - c)$ and let $J'$ be a fixed largest connected component of $H - c$. Moreover, let $J := H[J' \cup \{c\}]$ denote the

induced subgraph on all vertices of $J'$ plus $c$, and let $D := H - J'$. Note that by performing a disjoint union of the graphs $J$ and $D$ and identifying the vertex $c$ in $J$ with the vertex $c$ in $D$, the resulting graph is $H$. The goal is to replace each edge of the Weighted Vertex Cover-instance by a distinct copy of $J$ and each vertex of the Weighted Vertex Cover-instance by a distinct copy of $D$. Note that since $H$ only depends on the graph property $\Pi$, $H$ has constant size.

**Construction.** We are now able to present the PLS-reduction from Weighted Vertex Cover/3-swap to Weighted Deletion to $\Pi$/3-swap. Let $I = (G = (V, E), \omega)$ be an instance of Weighted Vertex Cover/3-swap with a constant maximum degree. We describe how to construct an instance $I' = (G' = (V', E'), \omega')$ of Weighted Deletion to $\Pi$ in polynomial time and a polynomial-time computable solution-mapper $f$ from $I'$ to $I$ that preserves local optimality. As described above, we extend the graph $G$ by replacing edges and vertices of $G$ with copies of $J$ and $D$, respectively. More precisely, we initialize $G'$ as an edgeless copy of $G$. For each vertex $v \in V$, we add a copy of $D$ to $G$ and identify $v$ with the vertex $c$ of $D$. We call this newly added subgraph (including vertex $v$) the *$D$-gadget of the vertex $v$*. Next, we describe the edge-gadgets. To this end, recall that $H$ is a connected graph with at least two vertices. Hence, since $J$ consists of the largest connected component of $H - c$ plus vertex $c$, $c$ has at least one neighbor in $J$. Fix any such neighbor $c'$ of $c$ in $J$. For each edge $\{u, v\}$ of $G$, we add a copy of $J$ to $G'$ and identify $u$ with $c$ and $v$ with $c'$. We call this newly added subgraph (including vertices $u$ and $v$) the *$J$-gadget of the edge $\{u, v\}$*. This completes the construction of $G'$.

Intuitively, for each edge $\{u, v\}$ of $G$, each solution for $I'$ contains at least one vertex of i) the $D$-gadget of the vertex $v$, ii) the $D$-gadget of the vertex $u$, or iii) the $J$-gadget of the edge $\{u, v\}$. In the following, we define the weights of the vertices of $G'$ in such a way that a solution with at least one vertex of $V' \setminus V$ is guaranteed to not be a 3-optimal solution for $I'$. This then implies that each 3-optimal solution for $I'$ contains for each edge $\{u, v\} \in E$ at least one of $u$ or $v$ and is thus a vertex cover for $I$.

Next, we define the weight function $\omega'$ and the solution-mapper $f$. Let $Z := 1 + \sum_{v \in V} \omega(v)$. For each vertex $v \in V$, we set $\omega'(v) := \omega(v)$ and for each vertex $v' \in V' \setminus V$, we set $\omega'(v') := Z$. We define the solution-mapper as follows: For each solution $S$ for $I'$, we set $f(S) := S$ if $S \subseteq V$. Otherwise, that is, if $S$ contains at least one vertex of $V' \setminus V$, we set $f(S) := V$. This completes the construction.

First, we show that $f$ is in fact a solution-mapper, that is, for each solution $S$ for $I'$, $f(S)$ is a vertex cover for $I$. If $S$ contains at least one vertex of $V' \setminus V$, then $f(S) = V$ is a vertex cover for $I$. Hence, we only have to show that a solu-

tion $S \subseteq V$ for $I'$ is also a vertex cover for $I$.

**Lemma 7.15.** *Let $S \subseteq V$. If $S$ is a solution for $I'$, then $S$ is a vertex cover for $I$.*

*Proof.* We show the statement by contraposition. Let $S \subseteq V$ such that $S$ is not a vertex cover for $I$. We show that $S$ is not a solution for $I'$. Let $\{u, v\}$ be an edge of $E$ such that $S$ contains neither $u$ nor $v$. Since $S$ is not a vertex cover for $I$, such an edge $\{u, v\}$ exists. We show that $S$ is not a solution for $I'$. Since $S$ contains only vertices of $V$ and contains neither $u$ nor $v$, $G' - S$ contains all vertices of the $D$-gadget of the vertex $u$, the $D$-gadget of the vertex $v$, and the $J$-gadget of the edge $\{u, v\}$. Recall that identifying vertex $c$ of $J$ with vertex $c$ of $D$ yields a graph isomorphic to $H$. Hence, $G' - S$ contains an induced subgraph isomorphic to $H$, since either $u$ or $v$ are identified with the vertex $c$ of the $J$-gadget of the edge $\{u, v\}$ and each of the vertices $u$ and $v$ is identified with the vertex $c$ of its respective $D$-gadget. Since $\Pi$ is hereditary, this implies that $G' - S$ does not satisfy property $\Pi$. Hence, $S$ is not a solution for $I'$. □

Consequently, $f$ is a solution-mapper.

**Correctness.** It remains to show that if $S$ is a 3-optimal solution for $I'$, then $f(S)$ is a 3-optimal vertex cover for $I$. To this end, we show two additional statements. First, we show that each 3-optimal solution for $I'$ contains only vertices of $V$. Second, we show that each vertex cover for $I$ is a solution for $I'$.

**Lemma 7.16.** *Let $S$ be a 3-optimal solution for $I'$. Then, $S$ contains no vertex of $V' \setminus V$.*

*Proof.* Let $S$ be a solution for $I'$. We show that $S$ is not 3-optimal for $I'$ if $S$ contains a vertex of $V' \setminus V$. Let $w$ be a vertex of $V' \setminus V$ contained in $S$. By construction, each vertex of $V' \setminus V$ is contained either in the $D$-gadget of some vertex $v \in V$ or in the $J$-gadget of some edge $\{u, v\} \in E$. If $w$ is contained in the $D$-gadget of some vertex $v \in V$, let $u$ be an arbitrary vertex of $V$. Let $S' := (S \setminus \{w\}) \cup \{u, v\}$. We show that $W := S \oplus S'$ is a valid and improving 3-swap for $S$ for $I'$. By definition of $S'$, $W$ is a 3-swap. Moreover, $W$ is an improving 3-swap for $S$ in $I'$ since $\omega'(w) = Z > \omega(u) + \omega(v) = \omega'(u) + \omega'(v)$. It remains to show that $S'$ is a solution for $I'$. To this end, first note that $S'' := S \cup \{u, v\} = S' \cup \{w\}$ is a solution for $I'$ since $\Pi$ is a hereditary graph property. Since $S''$ is a solution for $I'$, each connected component of $G' - S''$ fulfills $\Pi$. Let $C$ be the connected component of $G' - S'$ that contains $w$. Since $S'' = S' \cup \{w\}$, each other connected component of $G' - S'$ is a connected component of $G' - S''$ and thus fulfills $\Pi$. By the fact

that each minimal forbidden induced subgraph of $\Pi$ is connected, it remains to show that $G'[C]$ fulfills $\Pi$. Since $\Pi$ is nontrivial, $G'[C]$ fulfills $\Pi$ if $w$ is the unique vertex of $C$. Hence, in the following, we assume that $C$ has size at least 2. To show that $G'[C]$ fulfills $\Pi$, we show that $\alpha(G'[C])$ is lexicographically smaller than $\alpha(H)$ which implies that $G'[C]$ fulfills $\Pi$ by definition of $H$. We distinguish two cases. In both cases, we show that $G'[C]$ is isomorphic to a proper induced subgraph of $H$.

**Case 1:** $w$ is contained in the $D$-gadget of the vertex $v$**.** Since $v$ is contained in $S'$ and each other vertex of the $D$-gadget of $v$ has only neighbors within this gadget in $G'$, $C$ contains only vertices of the $D$-gadget of the vertex $v$. Since $S'$ contains $v$, $G'[C]$ is isomorphic to a proper induced subgraph of $H$.

**Case 2:** $w$ is contained in the $J$-gadget of the edge $\{u, v\}$**.** Since $u$ and $v$ are contained in $S'$ and each other vertex of the $J$-gadget of $\{u, v\}$ has only neighbors within this gadget in $G'$, $C$ contains only vertices of the $J$-gadget of the vertex $\{u, v\}$. Since $S'$ contains both $u$ and $v$, $G'[C]$ is isomorphic to a proper induced subgraph of $H$.

In both cases, $G'[C]$ is isomorphic to a proper induced subgraph of $H$. By choice of $H$, this implies that $G'[C]$ fulfills $\Pi$ and thus $S'$ is a solution for $I'$. □

Note that Lemma 7.15 and Lemma 7.16 imply that each 3-optimal solution $S$ for $I'$ is a vertex cover for $I$ and contains only vertices of $V$.

Next, we show that each vertex cover for $I$ is a solution for $I'$. Since we followed the construction of Lewis and Yannakakis [115], the statement follows. For the sake of completeness, we provide a proof.

**Lemma 7.17.** *Let $S \subseteq V$. If $S$ is a vertex cover for $I$, then $S$ is a solution for $I'$.*

*Proof.* Let $S$ be a vertex cover for $I$. We show that $G' - S$ fulfills $\Pi$. Since each minimal forbidden induced subgraph of $\Pi$ is connected, we only have to show that each connected component of $G' - S$ fulfills $\Pi$. Recall that each vertex of $V' \setminus V$ is either a) contained in a $D$-gadget of some vertex $v \in V$ and has only neighbors within this $D$-gadget or b) contained in a $J$-gadget of some edge $\{u, v\} \in E$ and has only neighbors within this $J$-gadget. Moreover, in $G'$, each vertex $v \in V$ has only neighbors within the $D$-gadget of $v$ and neighbors within the $J$-gadget of $\{u, v\}$ for each neighbor $u$ of $v$ in $G$.

Hence, there are three types of connected components of $G' - S$. If a connected component $C$ of $G' - S$ contains no vertex of $V$, then $G'[C]$ is either i) a proper and induced subgraph of the $D$-gadget of some vertex $v \in S$ or ii) a proper induced subgraph of the $J$-gadget of some edge $\{u, v\} \in E$. Otherwise, iii) $C$ contains a vertex $v \in V \setminus S$. For the first two types, $G'[C]$ is a proper and induced subgraph

of $H$. Since each graph with a lexicographically smaller $\alpha$-sequence than $H$ fulfills $\Pi$, Lemma 7.14 implies that $G'[C]$ fulfills $\Pi$.

Hence, to show that $G'-S$ fulfills $\Pi$, it remains to show that the induced subgraph of $G'$ of each connected component of $G'-S$ of the third type fulfill $\Pi$. To this end, we further analyze the structure of such connected components. Let $C$ be a such a connected component of $G'-S$ and let $v$ be a vertex of $V$ contained in $C$. If $C$ has size one, then $G'[C]$ fulfills $\Pi$, since $\Pi$ is nontrivial. Hence, in the following, assume that $C$ has size at least 2. Since $S$ is a vertex cover for $I$, $v$ has in $G'$ only neighbors in the $D$-gadget of $v$ and in the $J$-gadget of the edge $\{u,v\}$ for each neighbor $u$ of $v$ in $G$, $v$ is the unique vertex of $C \cap V$. More precisely, since for each neighbor $u$ of $v$ in $G$, $u$ is contained in $S$, $G'[C]$ contains a proper induced subgraph of the $J$-gadget of the edge $\{u,v\}$. Moreover, $C$ contains all vertices of the $D$-gadget of the vertex $v$. All these subgraphs share vertex $v$. We show that $\alpha(G'[C]) \prec \alpha(H)$. By choice of $H$ this then implies that $G'[C]$ fulfills $\Pi$.

To this end, we show that $\text{comp}(G'[C]-v) \prec \alpha(H)$. Since $C$ contains all vertices of the $D$-gadget of the vertex $v$, each connected component of $D_v - v$ is a connected component in $G'[C]-v$, where $D_v$ denotes the $D$-gadget of the vertex $v$. Moreover, each other connected component of $G'[C]-v$ is an induced subgraph of $J_{\{u,v\}}-\{u,v\}$, where $J_{\{u,v\}}$ denotes the $J$-gadget of the edge $\{u,v\}$ for some neighbor $u$ of $v$ in $G$. Recall that the number of vertices in $J$ minus 1 is equal to the largest value $x$ of $\alpha(H)$. Hence, each connected component of $G'[C]-v$ besides the connected components of $D_v - v$ contains less than $x$ vertices. Moreover, recall that $c$ denotes a vertex of $H$ with $\alpha(H) = \text{comp}(H-c)$ such that the connected components of $H-c$ are the connected components of $D-c$ plus the connected components of $J-c$. Let $\ell$ denote the number of connected components of $H-c$ of size exactly $x$. Since $D_v - v$ has exactly $\ell - 1$ connected components of size at most $x$, $\text{comp}(G'[C]-v)$ contains strictly less entries of value $x$ than $\alpha(H)$. Since each $\alpha$-sequence is in nonincreasing order, $\alpha(G'[C]) \preceq \text{comp}(G'[C]-v) \prec \alpha(H)$ and thus by definition of $H$, $G'[C]$ fulfills $\Pi$.

Hence, each connected component of $G'-S$ fulfills $\Pi$ which implies that $G'-S$ fulfills $\Pi$ and that $S$ is a solution for $I'$. □

With these statements, we are now able to show that $f$ preserves local optimality.

**Lemma 7.18.** *The solution-mapper $f$ preserves local optimality.*

*Proof.* We show the statement by contraposition. That is, for each solution $S$ for $I'$ for which $f(S)$ is not a 3-optimal vertex cover for $I$, we show that $S$ is not a 3-optimal solution for $I'$. Let $S$ be a solution for $I'$ for which $f(S)$ is not a 3-optimal vertex

cover for $I$. If $S$ contains a vertex of $V' \setminus V$, then $S$ is not a 3-optimal solution for $I'$ due to Lemma 7.16. Hence, in the following, assume that $S$ contains only vertices of $V$. Since $S$ is a solution for $I'$, Lemma 7.15 implies that $S$ is a vertex cover for $I$ and that $f(S) = S$. Since we assume that $f(S)$ is not a 3-optimal vertex cover for $I$, there is an improving 3-swap $W$ for $S$ for $I$. Let $S' := S \oplus W$. Since $S'$ is a vertex cover for $I$, Lemma 7.17 implies that $S'$ is a solution for $I'$. Moreover, since for each vertex $v \in V$, $\omega(v) = \omega'(v)$ and both $S$ and $S'$ contain only vertices of $V$, $S'$ is a better solution than $S$ for $I'$. Consequently, $S$ is not a 3-optimal solution for $I'$. Concluding, for each 3-optimal solution $S$ for $I'$, $f(S)$ is a 3-optimal vertex cover for $I$. □

Hence, for each nontrivial hereditary graph property $\Pi$ where each minimal forbidden induced subgraph is connected, Weighted $\Pi$ Subgraph/3-swap and Weighted Deletion to $\Pi$/3-swap are PLS-hard even on graphs of constant maximum degree. This completes the proof of Theorem 7.13.

## 7.4 Hardness of Finding 3-Optimal Dominating Sets

In this section, we consider Weighted Dominating Set.

> Weighted Dominating Set
> **Input:** A graph $G = (V, E)$ and a vertex-weight function $\omega\colon V \to \mathbb{N}$.
> **Output:** A dominating set in $G$ of minimum total weight.

Note that Weighted Dominating Set is a subset-weight minimization problem. In the following, we derive PLS-hardness for Weighted Dominating Set/3-swap by presenting a PLS-reduction from Weighted Vertex Cover/3-swap. Note that there is no graph property $\Pi$ such that Weighted Dominating Set is equal to Weighted Deletion to $\Pi$. Hence, the result does not follow from Theorem 7.13.

We show that the classic Karp-reduction [97] from Vertex Cover to Dominating Set also gives a PLS-reduction from Weighted Vertex Cover/$k$-swap to Weighted Dominating Set/$k$-swap for $k \geq 3$.

**Theorem 7.19.** *Let $k \geq 3$. There is a* PLS*-reduction from* Weighted Vertex Cover*/$k$-swap to* Weighted Dominating Set*/$k$-swap where the maximum degree of the output graph is at most two times the maximum degree of the input graph.*

*Proof.* Let $I = (G = (V, E), \omega\colon V \to \mathbb{N})$ be an instance of Weighted Vertex Cover/$k$-swap. We describe how to obtain an instance $I' = (G' = (V', E'), \omega'\colon V' \to \mathbb{N})$ of Weighted Dominating Set/$k$-swap and a solution-mapper $f$ from $I'$ to $I$ that preserves local optimality and can be computed in polynomial time. We can assume that each vertex of $V$ is incident with at least one edge in $G$.

Initially, we set $G'$ to be a copy of $G$. Afterwards, we add for each edge $\{u, v\} \in E$ a new vertex $x_{\{u,v\}}$ and make $\{u, v, x_{\{u,v\}}\}$ a clique in $G'$. Let $Z := 1 + \sum_{v \in V} \omega(v)$. We set $\omega'(v) := \omega(v)$ if $v$ is a vertex of $V$ and $\omega'(v) := Z$ otherwise. Finally, the solution-mapper $f$ is defined as $f(S) := (S \cap V) \cup \{u, v \mid x_{\{u,v\}} \in S\}$. Note that for each dominating set $S$ for $G'$, $f(S)$ is in fact a vertex cover in $G$ since for each edge $\{u, v\}$ of $E$, $S$ contains at least one vertex of $\{u, v, x_{\{u,v\}}\}$ which implies that $f(S)$ contains at least one endpoint of the edge $\{u, v\}$.

Next, we show that if for a dominating set $S$ in $G'$, $f(S)$ is not a $k$-optimal vertex cover for $G$, then $S$ is not a $k$-optimal dominating set in $G'$.

To this end, we first show that a dominating set $S$ in $G'$ is not $k$-optimal if $S$ contains any vertex of $V' \setminus V$. Let $\{u, v\}$ be an edge of $E$ and let $x_{\{u,v\}}$ be a vertex of $S$. Since $x_{\{u,v\}}$ is only adjacent to $u$ and $v$ in $G'$ and all three vertices form a clique in $G'$, $S' := (S \setminus \{x_{\{u,v\}}\}) \cup \{u, v\}$ is a dominating set in $G'$ and is improving by the fact that $\omega'(x_{\{u,v\}}) = Z > \omega'(u) + \omega'(v)$. Hence, we assume in the following, that $S$ is a subset of $V$. Moreover, since $x_{\{u,v\}}$ is only adjacent to $u$ and $v$, each dominating set $S \subseteq V$ contains at least one endpoint of the edge $\{u, v\}$. Consequently, each dominating set $S \subseteq V$ in $G'$ is a vertex cover of $G$. On the other hand, a vertex cover $S$ of $G$ is a dominating set in $G'$ since for each vertex $v$ of $G'$, there is some edge $\{u, w\} \in E$ such that $\{u, w\} \subseteq N_{G'}[v]$. We conclude, a vertex set $S \subseteq V$ is a dominating set in $G'$ if and only if $S$ is a vertex cover of $G$. Hence, if there is an improving $k$-swap $W$ for $f(S)$ in $G$, then $W$ is an improving $k$-swap for $S$ in $G'$ since $\omega'(v) := \omega(v)$ for each $v \in V$. □

Due to Theorem 7.19 and Theorem 7.1, imply PLS-hardness for Weighted Dominating Set/7-swap, and Theorem 7.19 and Theorem 7.6 imply PLS-hardness for Weighted Dominating Set/3-swap.

**Corollary 7.20.** Weighted Dominating Set/7-*swap is* PLS-*complete on graphs of maximum degree at most* 12 *and* Weighted Dominating Set/3-*swap is* PLS-*complete on graphs of constant maximum degree.*

## 7.5 Finding Locally Optimal Solutions for some Restricted 3-Swaps

We now show that we can find locally optimal solutions in polynomial time for many subset-weight optimization problems if we restrict the allowed 3-swaps as follows: we either only allow swaps that add at most one vertex to the current solution or we allow only swaps that remove at most one vertex from the solution. These are exactly the $(1,2)$-swaps and $(2,1)$-swaps.

Recall that a $(k^{\text{in}}, k^{\text{out}})$-swap $W$ for a set $S$ fulfills $|W \setminus S| \leq k^{\text{in}}$ and $|W \cap S| \leq k^{\text{out}}$. That is, a $(k,1)$-swap may add up to $k$ elements to the solution and removes at most one element from the solution.

Further, recall that a subset-weight optimization problem $L$ consists of functions $U$, $f$, and $\omega$, where for each instance $I$ of $L$, $U(I)$ is the universe of $I$, $f(I,S)$ checks if $S$ is a solution for $I$, and $\omega(I,u)$ assigns a weight to each $u \in U(I)$. Moreover, the functions $U$, $f$, and $\omega$ are polynomial-time computable.

### 7.5.1 Algorithms for Subset-Weight Optimization Problems with Provided Initial Solution

We start by bounding the length of any longest sequence of solutions that are consecutively improving $(k,1)$-neighbors for a given instance of a subset-weight maximization problem.

**Theorem 7.21.** *Let $L$ be a subset-weight maximization problem, let $I$ be an instance of $L$, and let $k \in \mathbb{N}$. Moreover, let $(S_0, S_1, \ldots, S_x)$ be a sequence of consecutive improving $(k,1)$-neighbors in $I$ that are all solutions for $I$. Then, $x \in \mathcal{O}(n^3)$, where $n$ denotes the size of the universe $U(I)$ of $I$.*

In other words, by starting with any solution for $I$, one can find a $(k,1)$-optimal solution in $I$ in a polynomial number of improving steps by applying a simple hill-climbing algorithm, that is, by replacing the current solution by an improving $(k,1)$-neighbor in $I$ until reaching a locally optimal solution.

*Proof of Theorem 7.21.* We first show that for each $\ell \in [1,x]$, $|S_\ell|$ is never smaller than $|S_{\ell-1}|$. Let $\ell \in [1,x]$. Assume towards a contradiction that $|S_\ell| < |S_{\ell-1}|$. Then, there is some $u \in S_{\ell-1}$ such that $S_\ell = S_{\ell-1} \setminus \{u\}$ since $S_\ell$ is a $(k,1)$-neighbor of $S_{\ell-1}$ in $I$. Since the weight of $S_\ell$ is defined as the sum of weights of the elements of $S_\ell$ and each element $u \in U(I)$ has a positive weight $\omega(I,u)$, the total weight of $S_\ell$ is not larger than the total weight of $S_{\ell-1}$, a contradiction. As a consequence,

there are at most $y \leq n$ distinct indices $\ell_1, \ldots, \ell_y$ such that $|S_{\ell_i}|$ is strictly larger than $|S_{\ell_i-1}|$. To prove that $x \in \mathcal{O}(n^3)$, we now show that for each $\ell \in [1, x - n^2]$, there is some $z \in [\ell, \ell + n^2]$ where $|S_z|$ is larger than $|S_{\ell-1}|$. In other words, after at most $n^2$ improving swaps that do not increase the size of the solution, the next improving swap increases the size of the solution.

Let $\ell \in [1, x - n^2]$, let $\sigma = (u_1, \ldots, u_n)$ be a fixed nondecreasing order of the elements of $U(I)$ by weight, and let for each $j \in [\ell, \ell + n^2]$, $q_j := \sum_{u_i \in S_j} i$ denote the sum of the indices of $S_j$ in $\sigma$. Note that if $S_j$ and $S_{j-1}$ have the same size, then $S_j \oplus S_{j-1}$ is an improving $(1,1)$-swap for $S_{j-1}$. That is, $S_j = (S_{j-1} \cup \{u_2\}) \setminus \{u_1\}$ for two elements $u_1 \in U(I)$ and $u_2 \in U(I)$ where the weight of $u_2$ is larger than the weight of $u_1$. Hence, $q_j$ is larger than $q_{j-1}$ if $S_j$ and $S_{j-1}$ have the same size. Since $q_j$ can never exceed $n^2$, there is some $z \in [\ell, \ell + n^2]$ where the size of $S_z$ is larger than the size of $S_{\ell-1}$. We conclude that $x \in \mathcal{O}(n^3)$. □

We call a subset-weight optimization problem $L$ *initializable*, if for each instance $I$ of $L$, one can compute some feasible solution for $I$ with respect to $L$ in polynomial time. Note that by definition of PLS, for any neighborhood structure $\mathcal{N}$, $L/\mathcal{N}$ can only be contained in PLS if $L$ is initializable. Using Theorem 7.21, we can show that we can find for each initializable subset-weight maximization problem and each constant value $k$ a $(k,1)$-optimal solution in polynomial time. We then use this fact to show a polynomial-time algorithm for minimization problems with the $(1,k)$-swap neighborhood.

**Theorem 7.22.** *Let $I$ be an instance of an initializable subset-weight optimization problem $L$, and let $k \geq 0$ be a constant.*

- *If $L$ is a maximization problem, then one can compute in polynomial time a $(k,1)$-optimal solution for $I$.*
- *If $L$ is a minimization problem, then one can compute in polynomial time a $(1,k)$-optimal solution for $I$.*

*Proof.* We first show the statement for the case that $L$ is a maximization problem. Recall that $L$ consists of the polynomial-time computable functions $U$, $f$, and $\omega$.

Note that we can compute some feasible solution $S_0$ of $L$ in polynomial time since $L$ is initializable. Now, let $n := |U(I)|$ and observe that $n$ is bounded by a polynomial in $|I|$ since $U(I)$ can be computed in polynomial time. Since $f$ and $\omega$ can be computed in polynomial time, we can check for a given solution $S$ in $n^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time if there is an improving $(k+1)$-swap $W$ such that $W$ is a $(k,1)$-swap for $S$ by considering all subsets of size at most $k+1$ of $U(I)$. Note that this is a polynomial running

time since $k$ is a constant. Hence, we can determine whether a solution $S$ is $(k,1)$-optimal and, if this is not the case, replace $S$ by an improving $(k,1)$-neighbor, both in polynomial time. Let $S_x$ be an arbitrary $(k,1)$-optimal solution in $I$ which can be found this way. Moreover, let $(S_0, S_1, \ldots, S_x)$ be the sequence of consecutive improving $(k,1)$-neighbors in $I$ starting from $S_0$ that lead to $S_x$. Due to Theorem 7.21, $x \in \mathcal{O}(n^3)$, which implies that we can compute a $(k,1)$-optimal solution for $I$ in polynomial time.

The statement for minimization problems with $(1,k)$-swaps now follows from Observation 2.15. □

Since for maximization problems there is no improving swap that only removes elements from the solution and for minimization problems there is no improving swap that only adds elements to the solution, Theorem 7.22 implies the following.

**Corollary 7.23.** *Let $L$ be an initializable subset-weight optimization problem. Then, for each instance $I$ of $L$, one can compute a 2-optimal solution for $I$ in polynomial time.*

All specific problems that we considered so far are subset-weight optimization problems; the following summarizes the consequences of Theorem 7.22 for these problems.

**Corollary 7.24.** *Let $k$ be a constant. One can compute in polynomial time a $(k,1)$-optimal solution for* Weighted Independent Set *and* Weighted $\Pi$ Subgraph *with polynomial-time decidable $\Pi$ and one can compute in polynomial time a $(1,k)$-optimal solution for* Weighted Vertex Cover, Weighted Dominating Set, *and* Weighted Deletion to $\Pi$ *with polynomial-time decidable $\Pi$ .*

### 7.5.2 A General Greedy Algorithm for Hereditary Subset-Weight Optimization Problems

So far, we considered the $(k,1)$-swap neighborhood for maximization problems and the $(1,k)$-swap neighborhood for minimization problems. We now swap the combinations of neighborhoods and optimization goals, that is, we consider the $(1,k)$-swap neighborhood for maximization problems and the $(k,1)$-swap neighborhoods for minimization problems. To simplify the discussion, let us consider maximization problems for now.

The crux behind the proof for the $(k,1)$-swap neighborhood was that one could bound the number of improving steps because for maximization $(0,1)$-swaps are never improving and all other $(k,1)$-swaps do not decrease the size of the solution. With

the $(1, k)$-swap neighborhood we do not have such a monotone behavior: Removing two vertices and adding one with a larger weight may increase the weight and simultaneously decrease the size of the solution. Afterwards, it could be possible that one performs $(1, 0)$-swaps that increase the size of the solution. Thus, the size of the solution may oscillate with the $(1, k)$-swap neighborhood.

A consequence of this is that we could not show an algorithm for all subset-weight maximization problems. Instead, we show an algorithm for *hereditary* subset-weight optimization problems, a natural restriction of subset-weight optimization problems that covers all specific problems considered in this chapter. We show that a simple greedy algorithm can find locally optimal solutions for hereditary maximization problems with $(1, k)$-swap neighborhood and for hereditary minimization problems with $(k, 1)$-swap neighborhood.

**Definition 7.25.** A subset-weight optimization problem $L$ is *hereditary*

- if $L$ is a maximization problem and for each instance $I$ of $L$ and each solution $S$ for $I$, each set $S' \subseteq U(I)$ with $S' \subseteq S$ is a solution for $I$ or
- if $L$ is a minimization problem and for each instance $I$ of $L$ and each solution $S$ for $I$, each set $S' \subseteq U(I)$ with $S \subseteq S'$ is a solution for $I$.

As mentioned above, all the specific problems considered in this chapter are hereditary subset-weight optimization problems:

- Weighted Independent Set,
- Weighted Vertex Cover,
- Weighted Dominating Set, and
- Weighted Deletion to $\Pi$ and Weighted $\Pi$ Subgraph for each hereditary graph property $\Pi$ for which one can check in polynomial time whether a given graph fulfills $\Pi$.

**Theorem 7.26.** *Let $L$ be a hereditary subset-weight optimization problem and let $I$ be an instance of $L$ for which there is at least one feasible solution.*

- *If $L$ is a maximization problem, then one can compute in polynomial time a solution $S$ for $I$ which is $(1, k)$-optimal for every $k \in \mathbb{N}$.*
- *If $L$ is a minimization problem, then one can compute in polynomial time a solution $S$ for $I$ which is $(k, 1)$-optimal for every $k \in \mathbb{N}$.*

**Algorithm 1** Greedy algorithm for a hereditary subset-weight minimization problem $L$

```
1: Input Instance I of L with functions U, f, and ω
2: Output a (k, 1)-optimal solution for I for each k ∈ ℕ
3: σ ← order of the elements of U(I) sorted non-increasingly by their weight
4: S ← U(I)
5: for all v ∈ σ do
6:     if S \ {v} is a solution for I then
7:         S ← S \ {v}
   return S
```

Note that in contrast to Theorem 7.22, this includes even non-constant values of $k$.

*Proof of Theorem 7.26.* We first consider the case where $L$ is a minimization problem. Recall that $L$ consists of the polynomial-time computable functions $U$, $f$, and $\omega$. Let $I$ be an instance of $L$. Since $L$ is hereditary and there is at least one solution for $I$, $U(I)$ is a solution for $I$. The algorithm works as follows: First, order the elements of $U(I)$ non-increasingly by their weight and initialize $S := U(I)$. Next, iterate over the computed order $\sigma$. Let $u$ be the currently considered element in the order $\sigma$. If $S \setminus \{u\}$ is a solution for $I$, replace $S$ by $S \setminus \{u\}$. After the algorithm iterated over the whole order $\sigma$, return the set $S$. Note that $S$ is a solution for $I$ since $S$ was initialized as $U(I)$ and we only removed elements from $S$ if the resulting set was still a solution for $I$. The pseudocode of the algorithm is given in Algorithm 1.

Next, we show that $S$ is in fact $(k, 1)$-optimal for every $k \in \mathbb{N}$. Assume towards a contradiction that this is not the case. Let $k$ be the smallest integer such that there is an improving $(k, 1)$-neighbor $S'$ of $S$. Since $S'$ is improving, $S \setminus S'$ is nonempty. Moreover, since $S'$ is an improving $(k, 1)$-neighbor of $S$, $S \setminus S'$ consists of a single element $s^{\text{out}}$. Let $S^{\text{in}} := S' \setminus S$. Since $S'$ is improving, $\sum_{u \in S^{\text{in}}} \omega(I, u) < \omega(I, s^{\text{out}})$. Consequently, since $\omega(I, u) > 0$ for each element $u$ of $U(I)$, $\omega(I, u) < \omega(I, s^{\text{out}})$ for each element $u$ of $S^{\text{in}}$. Hence, while iterating over the ordering $\sigma$, the element $s^{\text{out}}$ was considered before any element of $S^{\text{in}}$ was considered. Let $\widetilde{S}$ be the solution for $I$ when element $s^{\text{out}}$ was considered during the iteration over the ordering $\sigma$. Since $s^{\text{out}}$ is contained in $S$, $s^{\text{out}}$ was not removed from $\widetilde{S}$. Thus, $\widetilde{S} \setminus \{s^{\text{out}}\}$ is not a solution for $I$. Since $\widetilde{S}$ and $S'$ agree on all elements that were considered before $s^{\text{out}}$ and $\widetilde{S} \setminus \{s^{\text{out}}\}$ contains all elements that are considered after $s^{\text{out}}$, $\widetilde{S} \setminus \{s^{\text{out}}\}$ is a superset of $S'$. Because $L$ is hereditary, this implies that $S'$ is not a solution for $I$, a contradiction.

It remains to show that the algorithm runs in polynomial time. Since $U(I)$ can

**Algorithm 2** Greedy algorithm for a hereditary subset-weight maximization problem $L$

```
1: Input Instance I of L with functions U, f, and ω
2: Output a (1, k)-optimal solution for I for each k ∈ ℕ
3: σ ← order of the elements of U(I) sorted non-increasingly by their weight
4: S ← ∅
5: for all v ∈ σ do
6:     if S ∪ {v} is a solution for I then
7:         S ← S ∪ {v}
   return S
```

be computed in polynomial time, $U(I)$ has polynomial size. Hence, the order $\sigma$ can be computed in polynomial time. Finally, while iterating over the order $\sigma$, checking whether $S \setminus \{u\}$ is a solution for $I$ can be done in polynomial time, since the function $f$ can be computed in polynomial time. Since this is done exactly once for each element of $\sigma$, the whole algorithm runs in polynomial time.

For maximization problems, we may derive the statement by reducing to the dual minimization problem (which is also hereditary) using Observation 2.15. Alternatively, one may use Algorithm 2 whose correctness proof is a direct adaptation of the proof for Algorithm 1 and thus omitted. □

In the remainder of this section, we show that for Weighted Independent Set and Weighted Dominating Set Algorithm 2 and Algorithm 1 we can adapt the respective algorithms so that they have running time $\mathcal{O}(n \cdot \log(n) + m)$.

First, we show the statement for Weighted Independent Set.

**Corollary 7.27.** *One can compute in $\mathcal{O}(n \cdot \log(n) + m)$ time an independent set which is $(1, k)$-optimal for every $k \in \mathbb{N}$.*

*Proof.* Let $I = (G = (V, E), \omega)$ be an instance of Weighted Independent Set. Following Algorithm 2, we have to perform two things in the stated running time:

- sort the vertices of $V$ non-increasingly by their weight,
- while iterating over the order $\sigma$, for each vertex $v \in S$, check whether $S \cup \{v\}$ is an independent set in $G$.

The first step can be performed in $\mathcal{O}(n \cdot \log(n))$ time. Moreover, the second step can be performed in $\mathcal{O}(|N(v)|)$ time for each vertex $v \in V$ which gives a total running time of $\mathcal{O}(n + m)$. Hence, for Weighted Independent Set, Algorithm 2 can be performed in $\mathcal{O}(n \cdot \log(n) + m)$ time. □

Second, we show the statement for Weighted Dominating Set.

**Corollary 7.28.** *One can compute in $\mathcal{O}(n \cdot \log(n) + m)$ time a dominating set which is $(k, 1)$-optimal for every $k \in \mathbb{N}$.*

*Proof.* Let $I = (G = (V, E), \omega)$ be an instance of Weighted Dominating Set. Following Algorithm 1, we have to perform two things in the stated running time:

- sort the vertices of $V$ non-increasingly by their weight,
- while iterating over the order $\sigma$, for each vertex $v \in S$, check wether $S \setminus \{v\}$ is still a dominating set for $G$.

The first step can be performed in $\mathcal{O}(n \cdot \log(n))$ time. In the following, we show that the second step can performed in $\mathcal{O}(|N(v)|)$ time for each vertex $v \in V$, by using an additional variable $d_v$ that stores the number of times vertex $v$ is dominated in the current solution.

Recall that the initial dominating set of the algorithm contains all vertices. Hence, for each vertex $v \in V$, we initialize the variable $d_v := |N(v)| + 1$. This can be done in total time $\mathcal{O}(n + m)$. Whenever we remove a vertex $v$ from the current solution, we decrement $d_u$ for each vertex $u \in N[v]$. Hence, whenever we consider a dominating set $S$ during the iteration over the order $\sigma$, for each vertex $v \in V$, $d_v$ stores the number of times vertex $v$ is dominated by vertices of $S$ in $G$. Note that for each vertex $v \in V$, the update of the variables $d_u$ for all vertices $u \in N[v]$ can be done in $\mathcal{O}(|N(v)|)$ time.

We can now determine wether $S \setminus \{v\}$ is a dominating set in $G$ in time $\mathcal{O}(|N(v)|)$ for each vertex $v \in V$, where $S$ is the current solution when considering vertex $v$. Since $S$ is a dominating set in $G$, this can be done by checking wether $d_u > 1$ for each vertex $u \in N[v]$.

Hence, for Weighted Dominating Set, Algorithm 1 can be performed in $\mathcal{O}(n \cdot \log(n) + m)$ time. □

In summary, the results of this section now relate to our previous hardness results as follows: If we allow only $(1, 2)$-swaps or only $(2, 1)$-swaps, then we can find locally optimal solutions for Weighted Independent Set, Weighted Π Subgraph, Weighted Deletion to Π, and Weighted Dominating Set in polynomial time. In contrast, if we simultaneously allow $(1, 2)$-swaps and $(2, 1)$-swaps, then we allow all 3-swaps and both problems are PLS-complete even on graphs of constant maximum degree.

## 7.6 Concluding Remarks

We have shown that a large number of natural weighted subset-weight optimization problems, including Weighted Independent Set and Weighted Dominating Set, are PLS-hard already for the 3-swap neighborhood. Moreover, we showed that locally optimal solutions for these problems can be found in polynomial time if (i) each swap can add at most one vertex to the solution or (ii) each swap can remove at most one vertex from the solution.

**Open questions.** From a theoretical point of view, one open topic is to determine the precise degree bounds that separate the polynomial-time solvable and PLS-complete cases for Weighted Independent Set and Weighted Dominating Set for $k$-swaps with small constant $k$-values. In particular, there seems to be a lot of room for improvement on the degree bound for Weighted Independent Set/3-swap, since the maximum degree of the constructed instance in our PLS-reduction is 3140. It seems unlikely that the reduction we presented can be modified to show PLS-hardness for graphs with much smaller maximum degree since to ensure the correctness of reduction, large cliques are required in the constructed instance. Hence, to significantly improve on the degree bound for Weighted Independent Set/3-swap, a different reduction idea might be necessary. Moreover, we left open some subsets of 3-swap neighborhoods. For example, the complexity of Weighted Independent Set is open when all 3-swaps are allowed except all swaps of size exactly 2. If there is an algorithm that finds locally optimal solutions for such restricted swaps in polynomial time, this algorithm has to use significantly different techniques than the ones we presented in Section 7.5. Also, the complexity of many subset-weight optimization problems with respect to $k$-swap neighborhoods remains open. A prominent example that is not covered by our hardness results is Longest Path, where the goal is to find a path of maximum total weight. Note that Longest Path is a subset-weight optimization problem, where the universe consists of all edges of the input graph.

A further more general direction for subset-weight optimization problems would be to study the influence of the distribution of the weights on the complexity of the local search problems. For example, if the universe has size $n$ and the number of elements whose weight exceeds $n$ is constant, then a standard hill-climbing algorithm finds a locally optimal solution after $n^{\mathcal{O}(1)}$ improvement steps. Can we improve substantially on this observation?

Moreover, it would be interesting to further analyze the complexity of finding locally optimal solutions for local search problems with scalable $k$-neighborhoods for

small constant values of $k$. For example, one could consider Weighted Cluster Deletion or Weighted Cluster Editing with respect to the $k$-move neighborhood described in Chapter 5. In particular, it might be worth analyzing whether Weighted Cluster Deletion is PLS-hard with respect to the 1-move neighborhood, since this neighborhood describes the most simple change between any two solutions to the problem.

From a practical point of view, our hardness results are informative in the sense that efficient heuristics which employ the practically relevant swap neighborhoods must address the problem of potentially long sequences of improving swaps, which seems to be unavoidable when the aim is to find locally optimal solutions. One way to counter this is to restrict the neighborhoods further, like for example considering only $(1, 2)$-swaps or $(2, 1)$-swaps instead of all possible 3-swaps. Another option could be to use gap-variants of local search that we introduced in Chapter 3. Recall that in such a gap-version, we are not looking for any improving solution but only for those that improve the objective value by at least some threshold $d$. Clearly, the thresholds may be set sufficiently high so that any series of improvements must be bounded by a polynomial in the input size. However, setting the threshold too high may lead to a deterioration of solution quality. For practical purposes, it will thus be interesting to study the impact of the thresholds on the running time and solution quality. From a theoretical point of view, the introduction of improvement thresholds defines a new type of neighborhoods and it would be interesting to study for which thresholds these neighborhoods are tractable and for which they lead to PLS-hard local search problems.

# Chapter 8

# Conclusion

In this work, we considered the complexity of two main aspects of hill-climbing local search algorithms for several important optimization problems with respect to their arguably most natural scalable local neighborhoods. The questions we considered were Question 1: "How fast can we determine whether a given solution is locally optimal, and provide a better solution in the local neighborhood, if this is not the case?" and Question 2: "How fast can we find a locally optimal solution for a given problem instance?" In the following we provide a high-level description of our findings and propose some general open questions for future research.

## 8.1 "How Fast Can We Decide Whether a Given Solution is Locally Optimal?"

We considered Question 1 for local search versions of the optimization problem: Weighted Vertex Cover, Weighted Independent Set, Max $c$-Cut, Cluster Deletion, Cluster Editing, and Maximum Parsimony. On the positive side, we were able to present algorithms for each problem with a running time of the form $\ell^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time for structural parameters $\ell \leq |I|$. As we propose, for algorithms that search for a better solution in the scalable local neighborhood, such FPT running times are desirable over general FPT running times ($f(k+\ell) \cdot n^{\mathcal{O}(1)}$ time for an arbitrary computable function $f$), since the search radius $k$ can be assumed to be a small constant set by a user. On the negative side, we showed that for all considered problems the corresponding local search problem, with respect to the

most popular scalable local neighborhoods, is W[1]-hard when parameterized by $k$[1] and cannot be solved in $f(k) \cdot |I|^{o(k)}$ time for any computable function $f$, unless the Exponential Time Hypothesis fails. An exception is the less popular scalable neighborhood for Maximum Parsimony based on sECR-operations, for which we provided an FPT-algorithm with respect to the search radius $k$ (see Theorem 6.23).

**Outlook.** From both a practical and a theoretical point of view, there are several interesting potential future directions. As we showed for LS Max $c$-Cut in Chapter 4, and as was shown for Weighted Vertex Cover by Ullmann [164], hill-climbing algorithms based on scalable neighborhoods perform nicely as post-processing for state-of-the-art heuristics for the considered problems. In future work, it would be interesting to further analyze the usefulness of such hill-climbing algorithms as post-processing for other important optimization problems. For example, one might revisit $k$-Flip Max Sat, which was first analyzed by Szeider [161]. As we discussed in Section 2.8, the results of Szeider [161] together with Theorem 2.18 imply that $k$-Flip Max Sat can be solved in $(pq)^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time, where $q$ denotes the size of a largest clause and $p$ denotes the maximal number of occurrences of any variable. Note that the super-polynomial factor of the running time grows strongly only with the search radius $k$. Hence, for instances where both $p$ and $q$ are small, one might be able to solve $k$-Flip Max Sat for moderate values of $k$ efficiently. This motivates to experimentally evaluate the performance of a hill-climbing algorithm based on this scalable neighborhood as post-processing for state-of-the-art heuristics for Max Sat. Additionally, one might consider other parameters $\ell$ for which one tries to find algorithms that solve $k$-Flip Max Sat in $\ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. For example, one might look into parameters that measure how "difficult" finding an optimal solution for the Max Sat instance is. An example for such a measurement is the size of a minimal backdoor set [134, 175]. Roughly speaking, a backdoor set is a subset $X$ of the variables of the input formula $F$, such that for each assignment $\tau$ of $X$, the restriction of $F$ to $\tau$ belongs to some base class of formulas for which an optimal solution can be found in polynomial time.

Generally, it would also be interesting to evaluate the performance of stand-alone hill-climbing algorithms based on scalable neighborhoods. To do so, it might be beneficial to include aspects of iterated local search (ILS) in such algorithms. In particular, one might apply some random perturbation to the current solution, if searching the current local neighborhood seems to be to inefficient. Such a random perturbation during the hill-climbing algorithms is a *perturbation step*. For example,

[1]For Weighted Vertex Cover and Weighted Independent Set, these W[1]-hardness results were already known [52].

ILS-based heuristics for Max $c$-Cut usually use a probability for their perturbation step which is 1 if $k$ is at least 3 [120,178]. In other words, these ILS-based heuristics for Max $c$-Cut may perform some perturbation step while searching for 2-optimal solutions, but are guaranteed to perform some perturbation step, if the found solution is 2-optimal. As we showed experimentally, even $k$-optimal solutions for $k \leq 6$ can be found efficiently. One might thus consider ILS approaches that allow $k$ to grow arbitrarily large while letting the probability for a perturbation step increase, the larger $k$ gets. For example, whenever the search radius $k$ is incremented, with a probability of $1 - \frac{1}{2^k}$, one might perform a perturbation step and reset $k$ to 1. It would be interesting to combine our algorithm for LS Max $c$-Cut with such an ILS approach in the future and evaluate its performance in comparison to other heuristic approaches for Max $c$-Cut. Surely, this approach is not limited to LS Max $c$-Cut and might also be considered for other implementations of parameterized local search algorithms, such as Weighted Vertex Cover and Cluster Editing.

From a theoretical point of view, one might consider searching for a better solution in the $k$-neighborhood from the perspective of kernelization. Kernelization is a general concept from parameterized complexity theory that is closely related to FPT-algorithms [35, 44]. Roughly speaking, kernelization is the reduction of the input instance (in our case, the local search problem) into an an equivalent instance for which the whole size is bounded by some function of the parameter. This is mostly achieved by data reduction rules that remove easy parts from the instance, so that only the hard kernel remains. Formally, we say that a decision problem $L$ admits a kernel for some parameter $\ell$, if there is a polynomial-time algorithm $A$ that transforms each instance $I$ of $L$ into an equivalent instance $I'$ of $L$ of size at most $f(\ell)$. Such an algorithm $A$ is called a *kernelization algorithm* and $f(\ell)$ is referred to as the *size* of the *kernel* $I'$. If $f(\ell) \in \ell^{\mathcal{O}(1)}$, we say that $I'$ is a *polynomial kernel*. It is known that a decidable problem $L$ is in FPT with respect to some parameter $\ell$ if and only if $L$ admits a kernel for $\ell$ [35, 44]. Since a kernel of small size can be solved efficiently, a clear future goal is to analyze whether the considered local search problems of Chapters 3 to 6 admit polynomial kernels when parameterized by the parameter combinations for which we presented FPT-algorithms. Similar to the difficulty difference between strict and permissive local search for the considered local search problems, finding polynomial kernel for the strict version of parameterized local search problems might be impossible, even though any better solution can be found efficiently. Hence, we propose to introduce and analyze a relaxed version of kernelization for local search problems: *permissive kernelization*. One way to model permissive kernelization could be to allow the algorithm to (i) output a classical kernel or (ii) output any better solution. In future research it might be interesting to

analyze for which local search problems and which parameters permissive polynomial kernels are possible whereas classical polynomial kernels are not. From a practical point of view it would also be interesting to analyze the impact of (permissive) kernelization on the derived hill-climbing algorithms.

## 8.2 "How Fast Can We Find a Locally Optimal Solution?"

Regarding Question 2, we considered a large class of subset-weight optimization problems with respect to the $k$-swap neighborhood for $k \in \{2, 3\}$. One of these considered problems was Weighted Independent Set, for which we showed that finding a locally optimal solution is PLS-hard with respect to the 3-swap neighborhood even on graphs of constant maximum degree. This hardness result was then lifted to a large class of subset-weight optimization problems. From the positive side, we presented generic algorithms for many subset-weight optimization problems that find locally optimal solution in polynomial time when considering only the 2-swap neighborhood. Recall that for the problem considered in Chapter 4, namely Max Cut, finding a locally optimal solution was already shown to be PLS-complete with respect to the 1-flip neighborhood [47].

**Outlook.** A potential research direction for the future might be to lift the complexity class PLS to the realm of parameterized complexity. Recall that PLS contains those local search problems for which (i) an initial solution can be found in polynomial time and (ii) the local neighborhood of each solution can be searched in polynomial time. Moreover, recall that there is no known algorithm that finds a locally optimal solution for any PLS-complete problem in polynomial time. Hence, intuitively, the subclass of PLS-complete local search problems can be seen as an analogue of the class of NP-complete decision problems. Based on this interpretation, instead of asking whether a locally optimal solution can be found in polynomial time, one might be interested in asking whether a locally optimal solution can be found in FPT-time with respect to some input-dependent parameter $p$. In other words, is there a constant $c$ such that for each constant parameter value of $p$, one can find a locally optimal solution in $\mathcal{O}(n^c)$ time? This would give rise to a parameterized complexity class for local search problems: FPT-PLS. Note that for optimization problems $L$, where finding an optimal solution can be done in $f(p) \cdot n^{\mathcal{O}(1)}$ time for some parameter $p$, for each local neighborhood $\mathcal{N}$, $L/\mathcal{N}$ is contained in FPT-PLS with respect to $p$. This is true, since each globally optimal solution is locally optimal with respect to every lo-

cal neighborhood. For example, an optimal solution for Weighted Independent Set can be found in FPT-time with respect to the vertex cover number of the input graph. Hence, asking whether a local search problem $L/\mathcal{N}$ is contained in FPT-PLS with respect to $p$ is only of interest when considering a parameter $p$, for which finding an optimal solution for $L$ can presumably not be done in FPT-time with respect to $p$. Another extreme is when $L/\mathcal{N}$ is PLS-hard even on instances where $p$ is a constant. Then, $L/\mathcal{N}$ is not contained in FPT-PLS with respect to $p$, unless every PLS-hard problem can be solved in polynomial time. Note that the PLS-hardness of Max Cut/flip on graphs of constant maximum degree [47] and the PLS-hardness results for Weighted Independent Set/3-swap on graphs of constant maximum degree presented in Chapter 7 imply that both these problems are presumably not contained in FPT-PLS with respect to the maximum degree of the input graph. Hence, asking for the containment in FPT-PLS is only interesting for parameters $p$, where

(i) an optimal solution for $L$ can presumably not be found in FPT-time with respect to $p$ and

(ii) $L/\mathcal{N}$ is not PLS-hard on instances where $p$ is a constant.

An example for a local search problem and a parameter $p$ fulfilling these properties is Weighted Independent Set/$\mathcal{N}$ with respect to $p$, where $p$ denotes the largest assigned weight and where $\mathcal{N}$ is an arbitrary polynomial-time computable local neighborhood. The desired properties hold, since (i) Weighted Independent Set is NP-hard even if all weights are equal to 1 and (ii) a basic hill-climbing algorithm for Weighted Independent Set/$\mathcal{N}$ can find a locally optimal solution after $\mathcal{O}(p \cdot n)$ improvement steps, which each take polynomial time by choice of $\mathcal{N}$. Hence, Weighted Independent Set/$\mathcal{N}$ is in FPT-PLS with respect to the largest assigned weight for each polynomial-time computable local neighborhood $\mathcal{N}$.

Another example for a parameterized local search problem that is contained in FPT-PLS is Max Circuit with respect to any polynomial-time computable local neighborhood. In Max Circuit, the input consists of a Boolean acyclic circuit with $n$ input gates, $m$ output gates, and a weight function in the output gates, and the goal is to find an assignment of the input gates that maximizes the total weight of satisfied output gates [95]. Note that the NP-complete SAT-problem is a special case of Max Circuit on circuit with only a single output gate. Hence, for $p$ being the number of output gates, a globally optimal solution for Max Circuit cannot be found in $f(p) \cdot |I|^{\mathcal{O}(1)}$ time, unless P = NP. Moreover, Max Circuit/1-flip is PLS-complete [95]. Hence, it is well-motivated to ask whether Max Circuit/1-flip is in FPT-PLS with respect to the number of output gates. In fact, for each polynomial-time computable local neighborhood, a basic hill-climbing algorithm for Max Cir-

CUIT finds a locally optimal solution in $2^p \cdot |I|^{\mathcal{O}(1)}$ time, since there are only $2^p$ possible configurations of satisfied output gates and in the process of the hill-climbing algorithm, we never encounter the same configuration twice, since we always search for strictly better solutions.

Similar to FPT-PLS, one could also define a complexity class XP-PLS, which contains all parameterized local search problems, for which one can find a locally optimal solution in $\mathcal{O}(n^{f(p)})$ time. Note that the current toolkit of parameterized complexity is not capable of giving evidence that a parameterized local search problem is contained in XP-PLS but not in FPT-PLS. That is, we cannot easily lift known W[1]-hardness results from decision problems to local search problem, since the latter are total search problems and guarantee that there is always a solution we can find. Therefore, it would be interesting to find parameterized local search problems that are contained in XP-PLS, but for which the containment in FPT-PLS seem unlikely. Based on such problems, one could then try to develop a completeness theory to prove that other parameterized local search problems are presumably also not contained in FPT-PLS. In other words, it would also be of interest to find an analogue to W[1]-hardness in the context of parameterized local search problems to better analyze for which parameterized local search problems we can find a locally optimal solution in FPT-time, and for which problems XP-time is the best we can hope for.

# Bibliography


[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the Current Clique Algorithms Are Optimal, so Is Valiant's Parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018. (Cited on pp. 27, 33)

[2] Faisal N. Abu-Khzam, Shaowei Cai, Judith Egan, Peter Shaw, and Kai Wang. Turbo-Charging Dominating Set with an FPT Subroutine: Further Improvements and Experimental Analysis. In *Proceedings of the 14th Annual Conference on Theory and Applications of Models of Computation (TAMC '17)*, volume 10185 of *Lecture Notes in Computer Science*, pages 59–70, 2017. (Cited on p. 62)

[3] Benjamin L. Allen and Mike Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5(1):1–15, 2001. (Cited on pp. 150, 154, 155, 156, 169, 170, 178)

[4] Josh Alman and Virginia Vassilevska Williams. A Refined Laser Method and Faster Matrix Multiplication. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '21)*, pages 522–539. SIAM, 2021. (Cited on p. 33)

[5] Diogo Vieira Andrade, Mauricio G. C. Resende, and Renato Fonseca F. Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547, 2012. (Cited on p. 3)

[6] Alexandre A. Andreatta and Celso C. Ribeiro. Heuristics for the phylogeny problem. *Journal of Heuristics*, 8(4):429–447, 2002. (Cited on p. 150)

[7] Konstantin Andreev and Harald Räcke. Balanced Graph Partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006. (Cited on p. 99)

[8] Manuel Aprile, Matthew Drescher, Samuel Fiorini, and Tony Huynh. A tight approximation algorithm for the cluster vertex deletion problem. *Mathematical Programming*, 197(2):1069–1091, 2023. (Cited on pp. 129, 133, 146)

[9] David Arbour, Drew Dimmery, and Anup B. Rao. Efficient Balanced Treatment Assignments for Experimentation. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS '21)*, volume 130 of *Proceedings of Machine Learning Research*, pages 3070–3078. PMLR, 2021. (Cited on p. 63)

[10] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. (Cited on p. 11)

[11] Emmanuel Arrighi, Niels Grüttemeier, Nils Morawietz, Frank Sommer, and Petra Wolf. Multi-parameter analysis of finding minors and subgraphs in edge-periodic temporal graphs. In *Proceedings of the 48th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '23)*, volume 13878 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 2023. (Cited on p. IV)

[12] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation Clustering. *Machine Learning*, 56(1-3):89–113, 2004. (Cited on p. 101)

[13] Valentin Bartier, Gabriel Bathie, Nicolas Bousquet, Marc Heinrich, Théo Pierron, and Ulysse Prieto. PACE Solver Description: $\mu$Solver - Heuristic Track. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC '21)*, volume 214 of *LIPIcs*, pages 33:1–33:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on p. 102)

[14] 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. 101)

[15] Una Benlic and Jin-Kao Hao. Breakout Local Search for the Max-Cut problem. *Engineering Applications of Artificial Intelligence*, 26(3):1162–1173, 2013. (Cited on p. 91)

[16] Daniel Berend and Tamir Tassa. Improved bounds on Bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, 30(2):185–205, 2010. (Cited on p. 132)

[17] Piotr Berman and Marek Karpinski. On Some Tighter Inapproximability Results (Extended Abstract). In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP '99)*, volume 1644 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 1999. (Cited on pp. 63, 64)

[18] Alexander Bille, Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. A Graph-Theoretic Formulation of Exploratory Blockmodeling. In *Proceedings of the 21st International Symposium on Experimental Algorithms (SEA '23)*, volume 265 of *LIPIcs*, pages 14:1–14:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. (Cited on p. III)

[19] Thomas Bläsius, Philipp Fischbeck, Lars Gottesbüren, Michael Hamann, Tobias Heuer, Jonas Spinner, Christopher Weyand, and Marcus Wilhelm. PACE Solver Description: KaPoCE: A Heuristic Cluster Editing Algorithm. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC '21)*, volume 214 of *LIPIcs*, pages 31:1–31:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on p. 102)

[20] Sebastian Böcker. A golden ratio parameterized algorithm for Cluster Editing. *Journal of Discrete Algorithms*, 16:79–89, 2012. (Cited on p. 101)

[21] Hans L. Bodlaender. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. (Cited on p. 43)

[22] Hans L. Bodlaender, Michael R. Fellows, and Tandy J. Warnow. Two Strikes Against Perfect Phylogeny. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*, volume 623 of *Lecture Notes in Computer Science*, pages 273–283. Springer, 1992. (Cited on p. 150)

[23] Édouard Bonnet, Yoichi Iwata, Bart M. P. Jansen, and Lukasz Kowalik. Fine-Grained Complexity of $k$-OPT in Bounded-Degree Graphs for Solving TSP. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA '19)*, volume 144 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited on pp. 7, 17)

[24] Flavia Bonomo, Guillermo Durán, and Mario Valencia-Pabon. Complexity of the cluster deletion problem on subclasses of chordal graphs. *Theoretical Computer Science*, 600:59–69, 2015. (Cited on p. 130)

[25] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On Modularity Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, 2008. (Cited on p. 148)

[26] David Bryant, John Tsang, Paul E. Kearney, and Ming Li. Computing the quartet distance between evolutionary trees. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*, pages 285–286. ACM/SIAM, 2000. (Cited on p. 182)

[27] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An Efficient Local Search Algorithm for Minimum Vertex Cover. *Journal of Artificial Intelligence Research*, 46:687–716, 2013. (Cited on p. 2)

[28] Amir Carmel, Noa Musa-Lempel, Dekel Tsur, and Michal Ziv-Ukelson. The worst case complexity of maximum parsimony. *Journal of Computational Biology*, 21(11):799–808, 2014. (Cited on pp. 149, 180)

[29] Arnaud Casteigts, Nils Morawietz, and Petra Wolf. Distance to Transitivity: New Parameters for Taming Reachability in Temporal Graphs. In *Proceedings of the 49th International Symposium on Mathematical Foundations of Computer Science (MFCS '24)*, volume 306 of *LIPIcs*, pages 36:1–36:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. (Cited on p. IV)

[30] Luigi L. Cavalli-Sforza and Anthony W. F. Edwards. Phylogenetic analysis. Models and estimation procedures. *American Journal of Human Genetics*, 19(3 Pt 1):233–257, 1967. (Cited on p. 180)

[31] Vaggos Chatziafratis, Mohammad Mahdian, and Sara Ahmadian. Maximizing Agreements for Ranking, Clustering and Hierarchical Clustering via MAX-CUT. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS '21)*, volume 130 of *Proceedings of Machine Learning Research*, pages 1657–1665. PMLR, 2021. (Cited on p. 63)

[32] Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In *Proceedings of the 47th Annual Symposium on Theory of Computing (STOC '15)*, pages 219–228. ACM, 2015. (Cited on p. 101)

[33] Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005. (Cited on p. 158)

[34] Jianer Chen and Jie Meng. A $2k$ kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012. (Cited on p. 101)

[35] 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. 12, 13, 14, 25, 35, 42, 67, 71, 72, 112, 116, 120, 121, 158, 225)

[36] Jakob Dahlum, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Accelerating Local Search for the Maximum Independent Set Problem. In *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA '16)*, volume 9685 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2016. (Cited on p. 3)

[37] Constantinos Daskalakis and Christos H. Papadimitriou. Continuous Local Search. In *Proceedings of the 22nd Annual Symposium on Discrete Algorithms (SODA '11)*, pages 790–804. SIAM, 2011. (Cited on p. 185)

[38] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. (Cited on p. 10)

[39] Alexander Dobler, Manuel Sorge, and Anaïs Villedieu. Turbocharging Heuristics for Weak Coloring Numbers. In *Proceedings of the 30th Annual European Symposium on Algorithms (ESA '22)*, volume 244 of *LIPIcs*, pages 44:1–44:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on p. 62)

[40] Srinivas Doddi, Madhav V. Marathe, Andy Mirzaian, Bernard M. E. Moret, and Binhai Zhu. Map Labeling and Its Generalizations. In *Proceedings of the 8th Annual Symposium on Discrete Algorithms (SODA '97)*, pages 148–157. ACM/SIAM, 1997. (Cited on p. 1)

[41] Yuanyuan Dong, Andrew V. Goldberg, Alexander Noe, Nikos Parotsidis, Mauricio G. C. Resende, and Quico Spaen. A Local Search Algorithm for Large Maximum Weight Independent Set Problems. In *Proceedings of the 30th Annual European Symposium on Algorithms (ESA '22)*, volume 244 of *LIPIcs*,

pages 45:1–45:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on p. 186)

[42] Marco Dorigo and Thomas Stützle. *Ant colony optimization*. MIT Press, 2004. (Cited on p. 2)

[43] Martin Dörnfelder, Jiong Guo, Christian Komusiewicz, and Mathias Weller. On the parameterized complexity of consensus clustering. *Theoretical Computer Science*, 542:71–82, 2014. (Cited on pp. 7, 17, 102)

[44] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. (Cited on pp. 13, 14, 25, 67, 71, 158, 225)

[45] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification and scene analysis*. John Wiley & Sons, 1973. (Cited on p. 148)

[46] Dominic Dumrauf and Tim Süß. On the Complexity of Local Search for Weighted Standard Set Problems. In *Proceedings of the 6th Conference on Computability in Europe (CiE 10)*, volume 6158 of *Lecture Notes in Computer Science*, pages 132–140. Springer, 2010. (Cited on pp. 7, 186)

[47] Robert Elsässer and Tobias Tscheuschner. Settling the Complexity of Local Max-Cut (Almost) Completely. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP '11)*, volume 6755 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2011. (Cited on pp. 7, 20, 186, 188, 226, 227)

[48] 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. 26)

[49] Thomas Erlebach, Nils Morawietz, Jakob T. Spooner, and Petra Wolf. A cop and robber game on edge-periodic temporal graphs. *Journal of Computer and System Sciences*, 144:103534, 2024. (Cited on p. III)

[50] Thomas Erlebach, Nils Morawietz, and Petra Wolf. Parameterized Algorithms for Multi-Label Periodic Temporal Graph Realization. In *Proceedings of the 3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND '24)*, volume 292 of *LIPIcs*, pages 12:1–12:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. (Cited on p. IV)

[51] George F. Estabrook, F. R. McMorris, and Christopher A. Meacham. Comparison of Undirected Phylogenetic Trees Based on Subtrees of Four Evolutionary Units. *Systematic Biology*, 34(2):193–200, 1985. (Cited on p. 182)

[52] Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, and Yngve Villanger. Local search: Is brute-force avoidable? *Journal of Computer and System Sciences*, 78(3):707–719, 2012. (Cited on pp. 7, 17, 25, 64, 77, 99, 224)

[53] Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Peter Shaw. Efficient Parameterized Preprocessing for Cluster Editing. In *Proceedings of the 16th International Symposium on Fundamentals of Computation Theory (FCT '07)*, volume 4639 of *Lecture Notes in Computer Science*, pages 312–321. Springer, 2007. (Cited on p. 101)

[54] Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates Sunderland, 2004. (Cited on p. 149)

[55] Pedro Felzenszwalb, Caroline Klivans, and Alice Paul. Clustering with Semidefinite Programming and Fixed Point Iteration. *Journal of Machine Learning Research*, 23(190):1–23, 2022. (Cited on p. 63)

[56] David Fernández-Baca and Jens Lagergren. A Polynomial-Time Algorithm for Near-Perfect Phylogeny. *SIAM Journal on Computing*, 32(5):1115–1127, 2003. (Cited on p. 150)

[57] Paola Festa, Panos M. Pardalos, Mauricio G. C. Resende, and Celso C. Ribeiro. Randomized heuristics for the Max-Cut problem. *Optimization Methods and Software*, 17(6):1033–1058, 2002. (Cited on pp. 64, 91)

[58] Walter M. Fitch. Toward defining the course of evolution: minimum change for a specific tree topology. *Systematic Biology*, 20(4):406–416, 1971. (Cited on pp. 149, 180, 181)

[59] Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Fast Local Search Algorithm for Weighted Feedback Arc Set in Tournaments. In *Proceedings of the 24th Annual AAAI Conference on Artificial Intelligence (AAAI '10)*. AAAI Press, 2010. (Cited on p. 17)

[60] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully Polynomial-Time Parameterized Computations for

Graphs and Matrices of Low Treewidth. *ACM Transactions on Algorithms*, 14(3):34:1–34:45, 2018. (Cited on p. 43)

[61] Les R. Foulds and Ronald L. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3(1):43–49, 1982. (Cited on p. 149)

[62] Alan M. Frieze and Mark Jerrum. Improved Approximation Algorithms for MAX $k$-CUT and MAX BISECTION. *Algorithmica*, 18(1):67–81, 1997. (Cited on pp. 63, 64)

[63] Ganeshkumar Ganapathy, Vijaya Ramachandran, and Tandy Warnow. On contract-and-refine transformations between phylogenetic trees. In *Proceedings of the 15th Annual Symposium on Discrete Algorithms (SODA '04)*, pages 900–909. SIAM, 2004. (Cited on pp. 150, 151, 154)

[64] Ganeshkumar Ganapathy, Vijaya Ramachandran, and Tandy J. Warnow. Better Hill-Climbing Searches for Parsimony. In *Proceedings of the 3rd International Workshop on Algorithms in Bioinformatics (WABI '03)*, volume 2812 of *Lecture Notes in Computer Science*, pages 245–258. Springer, 2003. (Cited on pp. 150, 151)

[65] 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. 11, 63)

[66] Jaroslav Garvardt, Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. Parameterized Local Search for Max $c$-Cut. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI '23)*, pages 5586–5594. ijcai.org, 2023. (Cited on pp. I, 16)

[67] Jaroslav Garvardt, Nils Morawietz, André Nichterlein, and Mathias Weller. Graph Clustering Problems Under the Lens of Parameterized Local Search. In *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC '23)*, volume 285 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. (Cited on p. II)

[68] Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julián Mestre, and Stefan Rümmele. Turbocharging Treewidth Heuristics. *Algorithmica*, 81(2):439–475, 2019. (Cited on pp. 7, 17, 62)

[69] Serge Gaspers, Eun Jung Kim, Sebastian Ordyniak, Saket Saurabh, and Stefan Szeider. Don't Be Strict in Local Search! In *Proceedings of the 26th Annual AAAI Conference on Artificial Intelligence (AAAI '12)*. AAAI Press, 2012. (Cited on pp. 5, 7, 17, 18, 25, 71)

[70] Martin Josef Geiger. PACE Solver Description: A Simplified Threshold Accepting Approach for the Cluster Editing Problem. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC '21)*, volume 214 of *LIPIcs*, pages 34:1–34:2. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on p. 102)

[71] Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao. Local Search for the Maximum Parsimony Problem. In *Proceedings of the 1st International Conference on Advances in Natural Computation (ICNC '05)*, volume 3612 of *Lecture Notes in Computer Science*, pages 678–683. Springer, 2005. (Cited on p. 150)

[72] Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao. Progressive tree neighborhood applied to the maximum parsimony problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(1):136–145, 2008. (Cited on p. 150)

[73] Pablo A Goloboff. Character optimization and calculation of tree lengths. *Cladistics*, 9(4):433–436, 1993. (Cited on p. 150)

[74] Pablo A. Goloboff. Analyzing Large Data Sets in Reasonable Times: Solutions for Composite Optima. *Cladistics*, 15(4):415–428, 1999. (Cited on pp. 150, 151)

[75] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-Modeled Data Clustering: Exact Algorithms for Clique Generation. *Theory of Computing Systems*, 38(4):373–392, 2005. (Cited on p. 101)

[76] Jens Gramm, Arfst Nickelsen, and Till Tantau. Fixed-Parameter Algorithms in Phylogenetics. *The Computer Journal*, 51(1):79–101, 2008. (Cited on p. 183)

[77] Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. Efficient Bayesian Network Structure Learning via Parameterized Local Search on Topological Orderings. In *Proceedings of the 35th Annual AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 12328–12335. AAAI Press, 2021. (Cited on pp. IV, 7, 17)

[78] Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. On the Parameterized Complexity of Polytree Learning. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI '21)*, pages 4221–4227. ijcai.org, 2021. (Cited on p. IV)

[79] Niels Grüttemeier, Christian Komusiewicz, Nils Morawietz, and Frank Sommer. String Factorizations Under Various Collision Constraints. In *Proceedings of the 31th 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. V)

[80] 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. IV)

[81] Jiong Guo. A More Effective Linear Kernelization for Cluster Editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009. (Cited on p. 101)

[82] Jiong Guo, Sepp Hartung, Rolf Niedermeier, and Ondrej Suchý. The Parameterized Complexity of Local Search for TSP, More Refined. *Algorithmica*, 67(1):89–110, 2013. (Cited on pp. 7, 17)

[83] Jiong Guo, Danny Hermelin, and Christian Komusiewicz. Local search for string problems: Brute-force is essentially optimal. *Theoretical Computer Science*, 525:30–41, 2014. (Cited on pp. 7, 17)

[84] Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized Complexity of Vertex Cover Variants. *Theory of Computing Systems*, 41(3):501–520, 2007. (Cited on p. 25)

[85] Maozu Guo, Jian-Fu Li, and Yang Liu. Improving the Efficiency of p-ECR Moves in Evolutionary Tree Search Methods Based on Maximum Likelihood by Neighbor Joining. In *Proceeding of the 2nd International Multi-Symposium of Computer and Computational Sciences (IMSCCS '07)*, pages 60–67. IEEE Computer Society, 2007. (Cited on p. 151)

[86] Michael T. Hallett and Catherine McCartin. A Faster FPT Algorithm for the Maximum Agreement Forest Problem. *Theory of Computing Systems*, 41(3):539–550, 2007. (Cited on p. 182)

[87] Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science*, 494:86–98, 2013. (Cited on pp. 7, 17)

[88] Emanuel Herrendorf, Christian Komusiewicz, Nils Morawietz, and Frank Sommer. On the Complexity of Community-Aware Network Sparsification. In *Proceedings of the 49th International Symposium on Mathematical Foundations of Computer Science (MFCS '24)*, volume 306 of *LIPIcs*, pages 60:1–60:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. (Cited on p. IV)

[89] Jana Holznigenkemper, Christian Komusiewicz, Nils Morawietz, and Bernhard Seeger. On the Complexity of Computing Time Series Medians Under the Move-Split-Merge Metric. In *Proceedings of the 48th International Symposium on Mathematical Foundations of Computer Science (MFCS '23)*, volume 272 of *LIPIcs*, pages 54:1–54:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. (Cited on pp. IV, 140)

[90] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004. (Cited on pp. 2, 4, 14)

[91] Weiran Huang, Liang Li, and Wei Chen. Partitioned Sampling of Public Opinions Based on Their Social Dynamics. In *Proceedings of the 31st Annual AAAI Conference on Artificial Intelligence (AAAI '17)*, pages 24–30. AAAI Press, 2017. (Cited on p. 63)

[92] Pavel Hubácek and Eylon Yogev. Hardness of Continuous Local Search: Query Complexity and Cryptographic Lower Bounds. *SIAM Journal on Computing*, 49(6):1128–1172, 2020. (Cited on p. 185)

[93] Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. (Cited on p. 12)

[94] Tommy R Jensen and Bjarne Toft. *Graph coloring problems*, volume 39. John Wiley & Sons, 2011. (Cited on p. 63)

[95] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How Easy is Local Search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988. (Cited on pp. 7, 19, 26, 185, 186, 227)

[96] Viggo Kann, Sanjeev Khanna, Jens Lagergren, and Alessandro Panconesi. On the Hardness of Approximating Max $k$-Cut and its Dual. *Chicago Journal of Theoretical Computer Science*, 1997(2), 1997. (Cited on p. 63)

[97] Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. (Cited on pp. 63, 158, 211)

[98] Maximilian Katzmann and Christian Komusiewicz. Systematic Exploration of Larger Local Search Neighborhoods for the Minimum Vertex Cover Problem. In *Proceedings of the 31st Annual AAAI Conference on Artificial Intelligence (AAAI '17)*, pages 846–852. AAAI Press, 2017. (Cited on pp. I, 3, 6, 7, 17, 25, 26, 45, 47, 48, 51, 52, 62)

[99] Leon Kellerhals, Tomohiro Koana, André Nichterlein, and Philipp Zschoche. The PACE 2021 Parameterized Algorithms and Computational Experiments Challenge: Cluster Editing. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC '21)*, volume 214 of *LIPIcs*, pages 26:1–26:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on pp. 6, 101, 102, 103)

[100] Brian W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970. (Cited on p. 186)

[101] Hartmut Klauck. On the Hardness of Global and Local Approximation. In *Proceedings of the 5th Scandinavian Workshop on Algorithm Theory (SWAT '96)*, volume 1097 of *Lecture Notes in Computer Science*, pages 88–99. Springer, 1996. (Cited on p. 186)

[102] Jon M. Kleinberg and Éva Tardos. *Algorithm design*. Addison-Wesley, 2006. (Cited on pp. 16, 63, 64)

[103] Gary A. Kochenberger, Jin-Kao Hao, Zhipeng Lü, Haibo Wang, and Fred W. Glover. Solving large scale Max Cut problems via tabu search. *Journal of Heuristics*, 19(4):565–571, 2013. (Cited on p. 98)

[104] Christian Komusiewicz, Simone Linz, Nils Morawietz, and Jannik Schestag. On the Complexity of Parameterized Local Search for the Maximum Parsimony Problem. In *Proceedings of the 34th Annual Symposium on Combinatorial*

*Pattern Matching (CPM '23)*, volume 259 of *LIPIcs*, pages 18:1–18:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. (Cited on p. II)

[105] Christian Komusiewicz and Nils Morawietz. Can Local Optimality Be Used for Efficient Data Reduction? In *Proceedings of the 12th International Conference on Algorithms and Complexity (CIAC '21)*, volume 12701 of *Lecture Notes in Computer Science*, pages 354–366. Springer, 2021. (Cited on p. III)

[106] Christian Komusiewicz and Nils Morawietz. Finding 3-Swap-Optimal Independent Sets and Dominating Sets Is Hard. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS '22)*, volume 241 of *LIPIcs*, pages 66:1–66:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on p. III)

[107] Christian Komusiewicz and Nils Morawietz. Parameterized Local Search for Vertex Cover: When Only the Search Radius Is Crucial. In *Proceedings of the 17th International Symposium on Parameterized and Exact Computation (IPEC '22)*, volume 249 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on p. I)

[108] Christian Komusiewicz and Frank Sommer. Enumerating connected induced subgraphs: Improved delay and experimental comparison. *Discrete Applied Mathematics*, 303:262–282, 2021. (Cited on pp. 23, 91)

[109] Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012. (Cited on p. 111)

[110] Athanasios L. Konstantinidis and Charis Papadopoulos. Cluster Deletion on Interval Graphs and Split Related Graphs. *Algorithmica*, 83(7):2018–2046, 2021. (Cited on p. 130)

[111] Mirko Krivánek and Jaroslav Morávek. NP-Hard Problems in Hierarchical-Tree Clustering. *Acta Informatica*, 23(3):311–323, 1986. (Cited on p. 101)

[112] Harold W. Kuhn. The Hungarian Method for the Assignment Problem. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 29–47. Springer, 2010. (Cited on p. 139)

[113] Manuel Laguna. Tabu Search. In *Handbook of Heuristics*, pages 741–758. Springer, 2018. (Cited on p. 6)

[114] Mark D. M. Leiserson, Diana Tatar, Lenore J. Cowen, and Benjamin J. Hescott. Inferring Mechanisms of Compensation from E-MAP and SGA Data Using Local Search Algorithms for Max Cut. *Journal of Computational Biology*, 18(11):1399–1409, 2011. (Cited on pp. 63, 64)

[115] 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. 129, 205, 209)

[116] Ruizhi Li, Shuli Hu, Shaowei Cai, Jian Gao, Yiyuan Wang, and Minghao Yin. NuMWVC: A novel local search for minimum weighted vertex cover problem. *Journal of the Operational Research Society*, 71(9):1498–1509, 2020. (Cited on pp. 2, 186)

[117] Shaohua Li, Marcin Pilipczuk, and Manuel Sorge. Cluster Editing Parameterized Above Modification-Disjoint $P_3$-Packings. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS '21)*, volume 187 of *LIPIcs*, pages 49:1–49:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on p. 101)

[118] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated Local Search. In *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 320–353. Kluwer / Springer, 2003. (Cited on p. 5)

[119] Junjie Luo, Hendrik Molter, André Nichterlein, and Rolf Niedermeier. Parameterized Dynamic Cluster Editing. *Algorithmica*, 83(1):1–44, 2021. (Cited on pp. 102, 104)

[120] Fuda Ma and Jin-Kao Hao. A multiple search operator heuristic for the max-k-cut problem. *Annals of Operations Research*, 248(1-2):365–403, 2017. (Cited on pp. 6, 64, 65, 91, 92, 93, 94, 225)

[121] Bodo Manthey, Nils Morawietz, Jesse van Rhijn, and Frank Sommer. Complexity of Local Search for Euclidean Clustering Problems. *CoRR*, abs/2312.14916, 2023. (Cited on pp. III, 186)

[122] Dániel Marx. Searching the $k$-change neighborhood for TSP is W[1]-hard. *Operations Research Letters*, 36(1):31–36, 2008. (Cited on pp. 7, 17)

[123] Dávid Matolcsi and Zoltán Lóránt Nagy. Generalised outerplanar Turán numbers and maximum number of $k$-vertex subtrees. *Discrete Applied Mathematics*, 307:115–124, 2022. (Cited on p. 179)

[124] Ross M. McConnell and Jeremy P. Spinrad. Linear-Time Modular Decomposition and Efficient Transitive Orientation of Comparability Graphs. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 536–545. ACM/SIAM, 1994. (Cited on pp. 35, 58, 59)

[125] Wil Michiels, Emile H. L. Aarts, and Jan H. M. Korst. *Theoretical aspects of local search*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2007. (Cited on pp. 7, 186)

[126] Nenad Mladenovic and Pierre Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997. (Cited on p. 5)

[127] Nils Morawietz, Carolin Rehs, and Mathias Weller. A Timecop's Work Is Harder Than You Think. In *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS '20)*, volume 170 of *LIPIcs*, pages 71:1–71:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited on p. III)

[128] Nils Morawietz and Petra Wolf. A Timecop's Chase Around the Table. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS '21)*, volume 202 of *LIPIcs*, pages 77:1–77:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on p. III)

[129] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985. (Cited on p. 33)

[130] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, 2004. (Cited on p. 148)

[131] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. (Cited on pp. 13, 42)

[132] Kevin C. Nixon. The parsimony ratchet, a new method for rapid parsimony analysis. *Cladistics*, 15(4):407–414, 1999. (Cited on p. 150)

[133] Bruno C. S. Nogueira, Rian G. S. Pinheiro, and Anand Subramanian. A hybrid iterated local search heuristic for the maximum weight independent set problem. *Optimization Letters*, 12(3):567–583, 2018. (Cited on p. 186)

[134] Sebastian Ordyniak, André Schidler, and Stefan Szeider. Backdoor DNFs. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI '21)*, pages 1403–1409. ijcai.org, 2021. (Cited on p. 224)

[135] Sebastian Ordyniak and Stefan Szeider. Algorithms and Complexity Results for Exact Bayesian Structure Learning. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI '10)*, pages 401–408. AUAI Press, 2010. (Cited on pp. 7, 17)

[136] Ibrahim H Osman and James P Kelly. Meta-heuristics: an overview. *Meta-heuristics: Theory and applications*, pages 1–21, 1996. (Cited on p. 2)

[137] Christos H. Papadimitriou. The Complexity of the Lin–Kernighan Heuristic for the Traveling Salesman Problem. *SIAM Journal on Computing*, 21(3):450–465, 1992. doi:10.1137/0221030. (Cited on p. 7)

[138] Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007. (Cited on p. 11)

[139] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982. (Cited on p. 14)

[140] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, Approximation, and Complexity Classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. (Cited on pp. 63, 64)

[141] Svatopluk Poljak. Integer Linear Programs and Local Search for Max-Cut. *SIAM Journal on Computing*, 24(4):822–839, 1995. (Cited on p. 186)

[142] Pablo D Rabinowicz and Jeffrey L Bennetzen. The maize genome as a model for efficient sequence analysis of large plant genomes. *Current Opinion in Plant Biology*, 9(2):149–156, 2006. (Cited on p. 183)

[143] Celso C. Ribeiro and Dalessandro Soares Vianna. A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure. *International Transactions in Operational Research*, 12(3):325–338, 2005. (Cited on p. 150)

[144] David F. Robinson. Comparison of labeled trees with valency three. *Journal of Combinatorial Theory, Series B*, 11(2):105–119, 1971. (Cited on p. 155)

[145] D.F. Robinson and L.R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1):131–147, 1981. (Cited on pp. 154, 155, 182)

[146] Peter Rossmanith. Simulated Annealing. In *Algorithms Unplugged*, pages 393–400. Springer, 2011. (Cited on p. 2)

[147] Peter Sanders and Christian Schulz. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Proceeding of the 12th International Symposium on Experimental Algorithms (SEA '13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 164–175. Springer, 2013. (Cited on p. 99)

[148] David Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, 1975. (Cited on p. 149)

[149] David Sankoff, Yvon Abel, and Jotun Hein. A tree· a window· a hill; generalization of nearest-neighbor interchange in phylogenetic optimization. *Journal of Classification*, 11(2):209–232, 1994. (Cited on pp. 151, 152, 154, 180)

[150] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. (Cited on p. 101)

[151] Alejandro A. Schäffer and Mihalis Yannakakis. Simple Local Search Problems That are Hard to Solve. *SIAM Journal on Computing*, 20(1):56–87, 1991. (Cited on pp. 7, 20, 64, 186)

[152] Ernst Schröder. Vier combinatorische Probleme. *Zeitschrift Mathematischer Physik*, 15:361–376, 1870. (Cited on p. 180)

[153] Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004. (Cited on p. 101)

[154] Shinichi Shimozono. Finding Optimal Subgraphs by Local Search. *Theoretical Computer Science*, 172(1-2):265–271, 1997. (Cited on pp. 7, 186)

[155] VP Shylo, F Glover, and IV Sergienko. Teams of global equilibrium search algorithms for solving the weighted maximum cut problem in parallel. *Cybernetics and Systems Analysis*, 51(1):16–24, 2015. (Cited on p. 91)

[156] Srinath Sridhar, Kedar Dhamdhere, Guy E. Blelloch, Eran Halperin, R. Ravi, and Russell Schwartz. Algorithms for efficient near-perfect phylogenetic tree reconstruction in Theory and Practice. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):561–571, 2007. (Cited on p. 150)

[157] Luca Pascal Staus, Christian Komusiewicz, Nils Morawietz, and Frank Sommer. Exact Algorithms for Group Closeness Centrality. In *Proceedings of the 2nd SIAM Conference on Applied and Computational Discrete Algorithms (ACDA '23)*, pages 1–12. SIAM, 2023. (Cited on p. IV)

[158] Thomas Stützle and Rubén Ruiz. Iterated Local Search. In *Handbook of Heuristics*, pages 579–605. Springer, 2018. (Cited on pp. 5, 6)

[159] Anand Prabhu Subramanian, Himanshu Gupta, Samir R. Das, and Jing Cao. Minimum Interference Channel Assignment in Multiradio Wireless Mesh Networks. *IEEE Transactions on Mobile Computing*, 7(12):1459–1473, 2008. (Cited on p. 63)

[160] Sylwester Swat. PACE Solver Description: CluES - a Heuristic Solver for the Cluster Editing Problem. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC '21)*, volume 214 of *LIPIcs*, pages 32:1–32:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on p. 102)

[161] Stefan Szeider. The parameterized complexity of $k$-flip local search for SAT and MAX SAT. *Discrete Optimization*, 8(1):139–145, 2011. (Cited on pp. 7, 16, 17, 24, 79, 80, 99, 183, 224)

[162] Dekel Tsur. Faster Parameterized Algorithm for Cluster Vertex Deletion. *Theory of Computing Systems*, 65(2):323–343, 2021. (Cited on pp. 132, 133)

[163] Esther Ulitzsch, Qiwei He, Vincent Ulitzsch, Hendrik Molter, André Nichterlein, Rolf Niedermeier, and Steffi Pohl. Combining clickstream analyses and graph-modeled data clustering for identifying common response processes. *Psychometrika*, 86(1):190–214, 2021. (Cited on p. 101)

[164] Felix Ullmann. Engineering a Local Search Solver for Weighted Vertex Cover. Bachelor's thesis, Philipps-Universität Marburg, 2023. (Cited on pp. 7, 17, 62, 98, 224)

[165] Steven van Dijk. *Genetic algorithms for map labeling*. PhD thesis, Utrecht University, Netherlands, 2001. (Cited on p. 2)

[166] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001. (Cited on p. 2)

[167] Stefan Voß, Silvano Martello, Ibrahim H Osman, and Catherine Roucairol. Meta-heuristics: Advances and trends in local search paradigms for optimization. 2012. (Cited on p. 2)

[168] Tjark Vredeveld and Jan Karel Lenstra. On local search for the generalized graph coloring problem. *Operations Research Letters*, 31(1):28–34, 2003. (Cited on pp. 63, 64)

[169] Jiří Šíma, Pekka Orponen, and Teemu Antti-Poika. Some Afterthoughts on Hopfield Networks. In *Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM '99)*, volume 1725 of *Lecture Notes in Computer Science*, pages 459–469. Springer, 1999. (Cited on p. 63)

[170] Jiahai Wang. An improved discrete Hopfield neural network for Max-Cut problems. *Neurocomputing*, 69(13-15):1665–1669, 2006. (Cited on p. 63)

[171] Yang Wang, Zhipeng Lü, Fred Glover, and Jin-Kao Hao. Probabilistic GRASP-Tabu search algorithms for the UBQP problem. *Computers & Operations Research*, 40(12):3100–3107, 2013. (Cited on p. 91)

[172] Kunihiro Wasa, Yusaku Kaneta, Takeaki Uno, and Hiroki Arimura. Constant Time Enumeration of Bounded-Size Subtrees in Trees and Its Application. In *Proceedings of the 18th Annual International Conference on Computing and Combinatorics (COCOON '12)*, volume 7434 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 2012. (Cited on p. 179)

[173] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, 2000. (Cited on p. 10)

[174] Chris Whidden and Frederick A. Matsen IV. Calculating the Unrooted Subtree Prune-and-Regraft Distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(3):898–911, 2019. (Cited on p. 182)

[175] Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors To Typical Case Complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI '03)*, pages 1173–1178. Morgan Kaufmann, 2003. (Cited on p. 224)

[176] Gerhard J. Woeginger. Space and Time Complexity of Exact Algorithms: Some Open Problems (Invited Talk). In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC '04)*, volume 3162 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 2004. (Cited on p. 33)

[177] Qinghua Wu, Yang Wang, and Zhipeng Lü. A tabu search based hybrid evolutionary algorithm for the max-cut problem. *Applied Soft Computing*, 34:827–837, 2015. (Cited on p. 98)

[178] Wenxing Zhu, Geng Lin, and M. Montaz Ali. Max-$k$-Cut by the Discrete Dynamic Convexized Method. *INFORMS Journal on Computing*, 25(1):27–40, 2013. (Cited on pp. 64, 91, 225)