Parameterized Local Search for Vertex Cover: When Only the Search Radius Is Crucial

Christian Komusiewicz
Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

Nils Morawietz
Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

Abstract

A k-swap W for a vertex cover S of a graph G is a vertex set of size at most k such that $S' = (S \setminus W) \cup (W \setminus S)$, the symmetric difference of S and W, is a vertex cover of G. If $|S'| < |S|$, then W is *improving*. In LS-Vertex Cover, one is given a vertex cover S of a graph G and wants to know if there is an improving k-swap for S in G. In applications of LS-Vertex Cover, k is a very small parameter that can be set by a user to determine the trade-off between running time and solution quality. Consequently, k can be considered to be a constant. Motivated by this and the fact that LS-Vertex Cover is W[1]-hard with respect to k, we aim for algorithms with running time $\ell^{f(k)} \cdot n^{\mathcal{O}(1)}$ where ℓ is a structural graph parameter upper-bounded by n. We say that such a running time *grows mildly with respect to ℓ and strongly with respect to k*. We obtain algorithms with such a running time for ℓ being the h-index of G, the treewidth of G, or the modular-width of G. In addition, we consider a novel parameter, the maximum degree over all quotient graphs in a modular decomposition of G. Moreover, we adapt these algorithms to the more general problem where each vertex is assigned a weight and where we want to find a d-improving k-swap, that is, a k-swap which decreases the weight of the vertex cover by at least d.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Computing methodologies → Discrete space search

Keywords and phrases Local Search, Structural parameterization, Fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.20

Funding *Nils Morawietz*: Supported by the Deutsche Forschungsgemeinschaft (DFG), project OPERAH, KO 3669/5-1.

1 Introduction

Local search is one of the most successful heuristic strategies to tackle hard optimization problems [6, 16, 20]. Consequently, understanding when local search yields good results and improving local search approaches is of utmost importance. In its easiest form, local search follows a hill-climbing approach on the space of feasible solutions of the optimization problem at hand. In this setting, one chooses some initial feasible solution and then iteratively replaces the current solution by a better one in its local neighborhood until reaching a local optimum, that is, a solution that has no better solution in its neighborhood. Intuitively, it is clear that the larger the local search neighborhood, the better the final solution will be. At the same time, searching a larger neighborhood takes longer. In particular, for hard optimization problems, the running time will be superpolynomial when the neighborhood is too large. As a consequence, there is a trade-off between running time and solution quality that is governed by the size of the local search neighborhood.

Parameterized local search offers a framework that may guide the design process for algorithms that attempt to search larger local neighborhoods. When applying parameterized local search to an optimization problem, the first step is to define a measure of distance

between solutions. The local search neighborhood of a solution is then the set of solutions within distance at most k. Here, k is an *operational* parameter that can be set by the user and that does not depend on the input data. The hope is now that the superpolynomial part of the running time for searching the local neighborhood depends mostly on k. More precisely, the ultimate goal of parameterized local search is to devise an algorithm that determines in $f(k) \cdot n^{\mathcal{O}(1)}$ time whether there exists a better solution within distance at most k of the current one. Often such a running time is not possible, since most local search problems turn out to be W[1]-hard with respect to the parameter k [5, 10, 15, 14, 21, 24].

For example, when applying local search to Vertex Cover, the set of feasible solutions of a graph $G = (V, E)$ is naturally defined as the collection of vertex covers of G, that is, vertex sets $S \subseteq V$ that cover all edges of the graph. The most obvious choice for a local search neighborhood is the k-swap neighborhood. Here, two vertex sets S and S' are k-swap neighbors if and only if $(S \setminus S') \cup (S' \setminus S)$ has size at most k. 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 [10]. Thus, at first it may seem unlikely that parameterized local search can be successfully applied to LS-Vertex Cover. 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 [10]. 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 $2^{\Delta(G) \cdot k}$. It is also known, however, that LS-Vertex Cover can be solved in time $\mathcal{O}(2^k \cdot (\Delta(G) - 1)^{k/2} \cdot k^3 \cdot n)$ [18]. In this running time only k appears in the exponent, while $\Delta(G)$ 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 Δ is not too small. In particular, the running time is polynomial for every fixed k. The practical usefulness of the algorithm with this worst-case running time bound was confirmed by experiments which showed that LS-Vertex Cover can be solved efficiently for k up to 25 [18].

In this work, we aim to find further algorithms for LS-Vertex Cover that achieve such running times which can be considered practical even though the superpolynomial running time part depends not only on the operational parameter k. Before describing our results, let us formalize the class of running time functions that we aim to achieve.

▶ **Definition 1.1.** *Let $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a function. We say that f* grows mildly with respect to ℓ 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 FPT-algorithms whose running time grows strongly only with respect to k and mildly with respect to some other parameters. In our opinion, the usefulness of this setting is not limited to local search problems. Instead, it may be useful whenever

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

Our Results. We provide FPT-algorithms for LS-Vertex Cover parameterized by k and several structural parameters of G. Besides k, we consider the treewidth of the input graph G, denoted by $\mathrm{tw}(G)$, the h-index of the input graph G, denoted by $h(G)$, the modular-width

of G, denoted by $\mathrm{mw}(G)$, and a novel parameter, the maximum degree over all quotient graphs in a minimum-width modular decomposition, denoted by $\Delta_{\mathrm{md}}(G)$. 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 where the input graph is vertex-weighted, the cost of a vertex cover is the sum of its vertex weights, and we search for a swap that improves the current solution by at least d for some input value d. Local search approaches for Weighted Vertex Cover have been studied from a more practical perspective which motivates our study of weighted variants of LS-VC. In addition, for weighted local search problems, there may be exponentially long chains of local improvements before one finds a local optimum [17] even for swaps of constant size [19]. Here, using a gap-variant of local search could reduce the number of necessary steps by increasing the improvement per step. 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 [9]. 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 [18] 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 operational parameter k.

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. For LS-Vertex Cover, we show that, for each bag of the tree decomposition, it is sufficient to consider intersections of size at most $\lceil \frac{k}{2} \rceil$ with potential improving swaps. This reduces the running time for LS-Vertex Cover to $\mathcal{O}((\mathrm{tw}(G)^{3\cdot\lceil \frac{k}{2} \rceil} + k^2) \cdot n)$. Hence, compared to the algorithm for Gap LS-Weighted Vertex Cover, we are able to consider swaps of double the size.

We then consider parameters that are related to modular decompositions. These parameters measure a different structural aspect, the similarity of neighborhoods in the graph, than treewidth or the degree-related parameterizations. In particular, they can be very small in dense graphs. For Gap LS-Weighted Vertex Cover we develop an FPT-algorithm with running time $\mathcal{O}(\mathrm{mw}(G)^k \cdot k \cdot (\mathrm{mw}(G)+k) \cdot n + m)$ based on dynamic programming on a modular decomposition, where $\mathrm{mw}(G)$ is the modular-width of G, the size of the largest vertex set of any quotient graph of the modular decomposition. We then show an improvement of this algorithm in terms of the structural parameter. More precisely, we show that when processing a node of the decomposition in the dynamic programming algorithm, one may apply a branching algorithm to determine how the swap interacts with the current node. Superficially, this branching algorithm resembles the one for $\Delta(G)$ and k but including the information computed for other nodes of the decomposition requires to combine the branching with Knapsack DP-algorithms. This gives an FPT-algorithm for the parameters $\Delta_{\mathrm{md}}(G)$ and k. Recall that $\Delta_{\mathrm{md}}(G)$ is the maximum degree over all quotient graphs of a modular decomposition of minimum width which is upper-bounded by $\mathrm{mw}(G)$. We believe that this novel parameter can be useful in further algorithmic applications of modular decompositions. We remark that the presented algorithm for $\Delta_{\mathrm{md}}(G)$ and k only solves the unweighted gap version of LS-VC; an extension to Gap LS-Weighted Vertex Cover seems possible but somewhat tedious.

In a second improvement, we show that instead of modular-width one can also obtain FPT-algorithms when using the smaller splitwidth; the results for this parameter are deferred to the full version of this article. Another candidate parameterization would be cliquewidth. We do not consider cliquewidth here, since there is no polynomial-time polynomial-factor approximation of cliquewidth and thus the desired type of FPT running times can only be achieved when a clique decomposition is given as input.

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

The proofs of statements marked with a (*) are deferred to a full version. For further details regarding parameterized algorithms, refer to the textbook of Cygan et al. [7].

2 Preliminaries

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, we denote with $\binom{A}{2} := \{\{a, b\} \mid a \in A, b \in A, a \neq b\}$ the collection of all size-two 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.

Graph Notation. An (undirected) graph $G = (V, E)$ consists of a set of vertices V and a set of edges $E \subseteq \binom{V}{2}$. 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. Moreover, 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 $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$. If G is clear from the context, we may omit the subscript.

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 β that maps each node $x \in \mathcal{V}$ to a graph $\beta(x)$. 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 ℓ 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 recursively defined as $V(\beta(\ell))$ for leaf nodes ℓ and 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 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 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 with w_1 and v_2 is not adjacent with 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 the largest vertex set of any quotient graph and the *modular-width* of a graph G is the minimal width of any modular decomposition of G denoted by $\mathrm{mw}(G)$.

The formal definition of treewidth and tree decompositions is deferred to the appendix.

Vertex Cover Local Search. 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. Let S be a vertex cover of G. A *k-swap*, for $k \in \mathbb{N}$, is a vertex set W of size at most k and W is said to be *valid for S in G* if $S \oplus W$ is also a vertex cover of G. For each valid swap W for S in G, both $W \cap S$ and $W \setminus S$ are independent sets and $N(W) \setminus S = N(W \cap S) \setminus S \subseteq W$. A swap W is *connected* if $G[W]$ is connected. Let $\omega : V \to \mathbb{N}$. For some $X \subseteq V$, we set $\omega(X) = \sum_{x \in X} \omega(x)$. The *improvement* of W is defined as $\alpha_\omega^S(W) := \omega(W \cap S) - \omega(W \setminus S)$. Moreover, W is *improving* if $\alpha_\omega^S(W) > 0$ and *d-improving* for some $d \in \mathbb{N}$ if $\alpha_\omega^S(W) \geq d$. If S or ω are clear from the context, we may omit them. In this work, we study the following local search problems for Vertex Cover.

LS-Weighted Vertex Cover (LS-WVC)
Input: A graph $G = (V, E)$, a weight function $\omega : V \to \mathbb{N}$, 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 $\omega : V \to \mathbb{N}$, 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 $d = 1$. Let $I = (G = (V, E), S, \omega, k, d)$ be an instance of GLS-WVC. We say that $W \subseteq V$ *is a solution for I*, if W is a valid d-improving k-swap for S in G.

3 Basic Observations and Lower Bounds

In this section, we first define swap-instances which are instances obtained from 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. 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. Then, swap further vertices to make the swap W valid, that is, to maintain that S is a vertex cover. To simplify the instance, the set $W' \supseteq W$ of swapped vertices is then removed from the instance. 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'. Formally, this reads as follows.

▶ **Definition 3.1.** *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS-WVC and let $W \subseteq V$ be a k-swap. Define $W' := W \cup (N(W) \setminus S)$. The instance*

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

with $G' := G - (N(W \cap S) \cup W')$, $k' := k - |W'|$, $d' := d - \alpha(W')$, and $\omega'(v) := \omega(v)$ for each $v \in V(G')$ is the swap-instance for I and W.

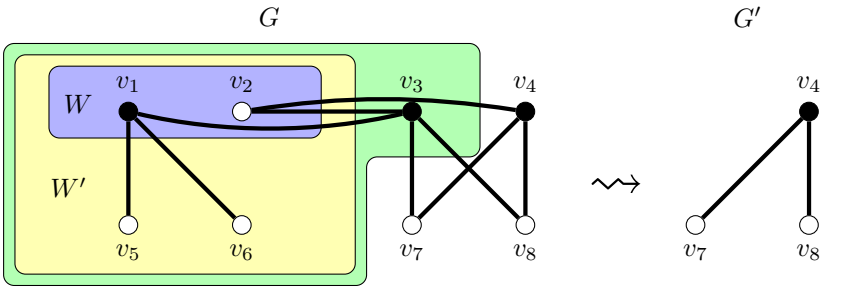

Figure 1 An instance $I := (G, S, k, d)$ (left) and the swap-instance $\text{swap}(I, W) := (G', S', k', d')$ (right) obtained from the swap $W := \{v_1, v_2\}$. The vertex cover vertices are black, the independent set vertices are white. The green area contains the vertices of $N(W \cap S) \cup W'$ 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_3 is not contained in G' since v_1 is adjacent to v_3 and leaves the vertex cover, which implies that v_3 cannot leave the vertex cover afterwards.

An example of a swap-instance can be seen in Figure 1. Note that W' is a subset of each valid swap W^* for S in G where $W \subseteq W^*$.

▶ **Lemma 3.2 (*).** *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS-WVC and let $W \subseteq V$ such that $W \cap S$ is an independent set. There is a solution W^* for I with $W \subseteq W^*$ if and only if $\text{swap}(I, W)$ is a yes-instance of GLS-WVC.*

If I is an instance of GLS-VC, then $k' + d' = k + d - 2 \cdot |W \cap S|$, since $\alpha(W') = -|W| + 2 \cdot |W \cap S|$. 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 show that it is sufficient to consider instances of GLS-VC where $k + d$ is even.

▶ **Lemma 3.3 (*).** *Let $I = (G, S, k, d)$ be an instance of GLS-VC where $k + d$ is odd. If I is a yes-instance of GLS-VC, then $I' := (G, S, k - 1, d)$ is a yes-instance of GLS-VC.*

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 3.4.** *Let $I = (G, \omega, S, k)$ be a yes-instance of LS-WVC. There is some valid improving k-swap W for S in G such that W is connected.*

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 potential improving swaps.

▶ **Lemma 3.5.** *Let $I = (G = (V, E), S, k, d)$ be an instance of GLS-VC, let W be a solution for I, and let $S_X := W \cap X \cap S$ and $C_X := W \cap X \setminus N[S_X]$ for some $X \subseteq V$. If $|S_X \cup C_X| > \frac{k+d}{2}$, then there is a solution 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 solution for I. Note that each d-improving k-swap contains at most $\frac{k-d}{2}$ vertices of $V \setminus S$. Since W is valid, it follows that $W^* \subseteq W$ and, thus, $|C_X| + |N(S_X) \setminus S| \leq \frac{k-d}{2}$. Moreover, since $|S_X \cup C_X| > \frac{k+d}{2}$, $|S_X| > d + \frac{k-d}{2} - |C_X| \geq d + |(N(S_X) \setminus S)|$. Hence, W^* is a solution for I.

If W^* is a proper subset of W, then the statement already holds. Hence, assume that $W = W^*$. As a consequence, $C_X = \emptyset$ and S_X has size more than $\frac{k+d}{2}$. Let S'_X be an arbitrary subset of S_X of size $\frac{k+d}{2}$ and let ℓ denote the size of the difference $S_X \setminus S'_X$. We show that $W' := S'_X \cup (N(S'_X) \setminus S)$ is a solution for I. Since S'_X is a subset of both S_X and S, and W^* is a valid swap, W' is a valid swap as well. Moreover, since W^* has size at most k and W' is a subset of W^*, W' is a k-swap. Finally, since $|S_X| = \frac{k+d}{2} + \ell$ and W^* is a k-swap, $W^* \setminus S_X = N(S_X) \setminus S$ has size at most $\frac{k-d}{2} - \ell$. Hence, W' is d-improving since $W' \setminus S'_X = N(S'_X) \setminus S$ is a subset of $N(S_X) \setminus S$ and thus has size at most $\frac{k-d}{2} - \ell$. ◀

To obtain linear FPT running times, we handle instances with small values of k separately.

▶ **Lemma 3.6 (*).** *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 $k+d \leq 4$.*

Lower Bounds. Let $\omega < 2.373$ be the matrix multiplication constant [3]. 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 [23]. It is a long-standing question whether this running time can be improved to $\mathcal{O}(n^{(\omega/3-\varepsilon)k})$ [1, 25]. Assuming that this is not the case, we obtain the following lower bounds for our considered problem.

▶ **Theorem 3.7.** *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 $\mathcal{O}(n^{(\omega/3-\varepsilon)k})$ time.*

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 $I^* = (G^* = (V, E^*), k)$ 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 $\ell := \max\{n' - \frac{k'-d}{2}, \Delta(G'), \mathrm{vc}(G'), |S|, \mathrm{mw}(G')\}$ is at most n. First, let $G = (V, E)$ be the complement graph of G^*, that is, $E := \binom{V}{2} \setminus E^*$. Note that a set $X \subseteq V$ is a clique in 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 in G of 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 all $v \in V^*$. 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 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 for G' such that $|S \oplus S'| = k'$ and $|S'| \leq |S| - d$. Consequently, I' is a yes-instance of GLS-VC.

($\Leftarrow$) Let $S' \subseteq V'$ such that $|S \oplus S'| \leq k'$ and $|S'| < |S| - d$. Consequently, $C := S \setminus S'$ is non-empty. We show that C is an independent set of size k in G. Since S' is a vertex cover for G' and every vertex of V^* is adjacent to every vertex of V, it follows that $V^* \subseteq S'$. By the fact that $|S \oplus S'| \leq 2k - d$, $S' \setminus S = V^*$, and V^* has size $k - d$, C 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 k. Moreover, since S' is a vertex cover for G', no two vertices of C are adjacent. Consequently, C is an independent set of size k in G and, thus, I^* is a yes-instance of Clique.

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

for G', $\mathrm{vc}(G') \leq n$. Next, we show that the modular-width of G' is at most n. Let $(\mathcal{T}_1, \beta_1)$ be a modular decomposition of G and let $(\mathcal{T}_2, \beta_2)$ be a modular decomposition of $G'[V' \setminus V]$. Since there is an edge between any pair of vertices of V and $V' \setminus V$, a modular decomposition $(\mathcal{T}, \beta)$ of G' can be obtained by combining $(\mathcal{T}_1, \beta_1)$ and $(\mathcal{T}_2, \beta_2)$ in the following way: We add a new root x^* where $\beta(x^*)$ is a graph consisting of a single edge and the vertices of $\beta(x^*)$ are the roots of the two modular decompositions $(\mathcal{T}_1, \beta_1)$ and $(\mathcal{T}_2, \beta_2)$. Note that $\mathrm{mw}(G) \leq n$. Moreover, since $V' \setminus V$ is an independent set, we have $\mathrm{mw}(G'[V' \setminus V]) = 2$. Hence, $\mathrm{mw}(G') \leq n$.

Now, if we have an algorithm A solving GLS-VC in $\mathcal{O}(\ell^{(\omega/3-\varepsilon)\cdot\frac{k+d}{2}})$ time for $\varepsilon > 0$, 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 3.8 (*).** *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 $\mathcal{O}(n^{(\omega/3-\varepsilon)k})$ time.*

4 Parameterization by Treewidth

In this section, we present FPT-algorithms for k and the treewidth of G.

Intuitively, the algorithms 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 $r + 1$ with an improving swap W of size at most k.

▶ **Theorem 4.1.** *Let $G = (V, E)$ be an undirected graph, let $\omega : V \to \mathbb{N}$ be a weight function, let $S \subseteq V$ be a vertex cover in G, and let k be a natural number. Given a nice tree decomposition of width r for G with $\mathcal{O}(n)$ bags, one can compute in $\mathcal{O}((r^k + k^2) \cdot n)$ time a valid k-swap W for S in G such that $\alpha(W)$ is maximal under all valid k-swaps for S in G.*

Proof. Due to Lemma 3.6, the statement holds for $k \leq 2$. In the following, we show the running time by describing a dynamic programming algorithm for $k \geq 3$.

Let $N_x(U) := N(U) \cap \beta(x)$ denote the neighbors of U in the bag of $x \in \mathcal{V}$. For a node $x \in \mathcal{V}$, we define with V_x the union of all bags $\beta(y)$, where y is reachable from x in $\mathcal{T}$. Moreover, we set $G_x := G[V_x]$ and $E_x := E_G(V_x)$. 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 $W_x := S_x \cup C_x \cup (N_x(S_x) \setminus S)$. Hence, $|S_x \cup C_x| \leq k$.

Each entry stores the maximal improvement $\alpha_S(W)$ 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 each valid swap containing S_x) exactly in C_x.

To ensure that we do not have to evaluate entries where $|S_x \cup C_x| > k$, we define for all $x \in \mathcal{V}, S_x \subseteq \beta(x) \cap S, C_x \subseteq \beta(x) \setminus S$, and $k' \in [0, k]$, $f_x(S_x, C_x, k') := D_x[S_x, C_x, k']$ if $|S_x \cup C_x| \leq k$ and $f_x(S_x, C_x, k') := -\infty$, otherwise.

For each leaf node ℓ of $\mathcal{T}$, we fill the table D_ℓ by setting $D_\ell[\emptyset, \emptyset, k'] := 0$ for each $k' \in [0, k]$.

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 give only an informal proof of the correctness; the formal proof is omitted.

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 remove v from the corresponding set (S_x or C_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 $\beta(x) \setminus \beta(y)$. The entries for x can be computed as follows:

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

where $W_x := S_x \cup C_x \cup (N_x(S_x) \setminus S)$ and $C^* := (N_x(v) \setminus S) \setminus N(S_x \setminus \{v\})$.

Informally, if v is a vertex of W_x, we have to consider the entry 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 $\mathcal{T}$. The entries for x can be computed as follows:

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

where $W_x := S_x \cup C_x \cup (N_x(S_x) \setminus S)$.

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 on both 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 $\alpha(W_x)$ 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, the corresponding swap can be found via traceback.

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 $\mathcal{O}(n)$ bags. 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 $x \in \mathcal{V}$, $\beta(x) \leq r+1$, we have that all dynamic programming tables together contain $\mathcal{O}(\binom{r+1}{\leq k} \cdot k \cdot n)$ entries, where $\binom{r+1}{\leq k}$ denotes the number of different subsets of size at most k of a set of size $r+1$. By the following claim, we can compute each of them efficiently.

▷ Claim (*). After a preprocessing running in $\mathcal{O}((\binom{r+1}{\leq k} \cdot k^2 + \binom{r+1}{\leq k-1} \cdot r \cdot k + r^2) \cdot n)$ time, one can compute each entry of each table D_x in $\mathcal{O}(k)$ time.

Note that there are $\mathcal{O}((\binom{r+1}{\leq k} \cdot k \cdot n)$ entries in total. Hence, the whole algorithm runs in $\mathcal{O}((\binom{r+1}{\leq k} \cdot k^2 \cdot n)$ time which is $\mathcal{O}(r^k \cdot n)$ time if $r \geq 2$ (the proof of this fact is deferred to Appendix B) and in $\mathcal{O}(2^r \cdot k^2 \cdot n) = \mathcal{O}(k^2 \cdot n)$ time, otherwise. ◀

The dynamic programming algorithm deviates from the simple idea mentioned above in the following detail: it considers only a) the intersection W_x^S of $W \cap S$ with the vertices of the current bag and b) 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 to $r^{\frac{k+d}{2}}$ for GLS-VC due to Lemma 3.5. 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 to $r^{\frac{k+1}{2}}$.

▶ **Theorem 4.2 (*).** *Let $I = (G = (V, E), S, k, d)$ be an instance of GLS-VC. Given a nice tree decomposition of width r for G with $\mathcal{O}(n)$ bags. One can solve I in $\mathcal{O}((r^{\frac{k+d}{2}} + k^2) \cdot n)$ time.*

Since computing a tree decomposition of minimal width is NP-hard, we cannot directly obtain a running time of $\mathcal{O}((\mathrm{tw}(G)^k + k^2) \cdot n)$ and $\mathcal{O}((\mathrm{tw}(G)^{\frac{k+d}{2}} + k^2) \cdot n)$, respectively. We can, however, compute a nice tree decomposition of width $\mathrm{tw}(G)$ in $\mathcal{O}(n+m)$ time if $\mathrm{tw}(G) \leq 1800$ [4]. Moreover, for each $r \geq 0$, one can compute a nice tree decomposition of G of width $1800 \cdot r^2$ or correctly output that $\mathrm{tw}(G) > r$ in $\mathcal{O}(r^7 \cdot n \cdot \log(n))$ time [11]. Hence, we can compute in $\mathcal{O}(\mathrm{tw}(G)^8 \cdot n \cdot \log(n))$ time [11] a nice tree decomposition of G of width at most $1800 \cdot \mathrm{tw}(G)^2$. 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.

▶ **Corollary 4.3.** *GLS-VC can be solved in $\mathcal{O}((\mathrm{tw}(G)^{\frac{3\cdot(k+d)}{2}} + k^2) \cdot n \cdot \log(n))$ time and GLS-WVC can be solved in $\mathcal{O}((\mathrm{tw}(G)^{3k} + k^2) \cdot n \cdot \log(n))$ 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 3.7, since $\mathrm{tw}(G) \leq \mathrm{vc}(G)$.

5 Degree-Related Parameterizations

In this section, we present FPT-algorithms for the parameters maximum degree $\Delta(G)$ and k and for the h-index of G and k. 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 5.1 (*).** *GLS-WVC can be solved in $\mathcal{O}(k \cdot \log(k) + n)$ time if $h(G) \leq 1$.*

Hence, from now on, we assume that $h(G)$ and $\Delta(G)$ are at least 2.

5.1 Parameterizing Unweighted Gap Local Search by Maximum Degree

The main result in this section for GLS-VC is the following.

▶ **Theorem 5.2 (*).** *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 to slightly adapt the known $\mathcal{O}(2^k \cdot (\Delta-1)^{\frac{k+1}{2}} \cdot k^2 \cdot n)$-time algorithm for LS-VC [18] 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, however, this is not the case: an improvement of at least 2 may be only achievable by swapping vertices that 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 one 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 5.3 (*).** *Let $I = (G = (V, E), S, k, d)$ be a yes-instance of GLS-VC and let v be a vertex of S with $N(v) \subseteq S$. There is a solution W for I such that either $v \in W$ or $|W \cap N(v)| \geq 2$.*

Hence, we obtain the following branching rule. Here, we are interested in swapping two independent neighbors of v at a time to obtain a better branching vector than the one, we could obtain by swapping only a single neighbor of v at a time.

▶ **Branching Rule 5.1.** *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-swap W_j for S in G if there is any. Consider some valid minimum solution W for I and let C be a connected component in $G[W]$ with the minimal improvement. Let $\ell := \alpha(C)$ and let $W' = (W \setminus C) \cup W_\ell$. Since W_ℓ contains at most $|C|$ vertices, we have $|W'| \leq k$. The resulting swap W' is a solution for I if $W \cap N[W_\ell] = \emptyset$.

The idea of the branching rule is now the following: either the d-improving swap contains W_ℓ, some neighbor of W_ℓ, or no connected component that is exactly ℓ-improving. First, we present an algorithm to efficiently find the swaps W_j for all $j \in [1, d]$ using the algorithm of Katzmann and Komusiewicz [18] as a subroutine and afterwards, we formally prove the correctness of this branching.

▶ **Proposition 5.4 (*).** *Let $I = (G = (V, E), S, k, d)$ be an instance of GLS-VC. For all $j \in [1, d]$, one can find some minimum valid connected j-improving k-swap W_j for S in G with $|W_j \setminus S| \leq (k-d)/2$ or correctly output that no such swap exists in total in $\mathcal{O}(2^k \cdot (\Delta-1)^{(k+d)/2} \cdot k^3 \cdot n)$ time.*

▶ **Proposition 5.5.** *Let $I = (G = (V, E), S, k, d)$ be a yes-instance of GLS-VC and, for each $j \in [1, \lfloor \frac{d}{2} \rfloor]$, let W_j denote a minimum valid connected j-improving $(k-d+j)$-swap for I. There is a solution 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 solution for I. Hence, the improvement of W is exactly d. 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 $\alpha(C)$, that is, the connected swap of W with the smallest improvement. Let $\ell := \alpha(C)$. Since W is not connected and has improvement exactly d, $\ell \leq \lfloor \frac{d}{2} \rfloor$. Note that $\ell \geq 1$ as, otherwise, W is not minimum. Moreover, 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, $|W \setminus S| \leq (k-d)/2$, which implies that $|C| \leq k - d + \ell$. Recall that W_ℓ is some minimum valid connected ℓ-improving $(k-d+\ell)$-swap for I. Hence, $|C| \geq |W_\ell|$. Recall that by assumption $W_\ell \not\subseteq W$ and $N(W_\ell) \cap W = \emptyset$. Since W is minimum, this implies that $W_\ell \cap W = \emptyset$, as otherwise $W_\ell \cap W$ is a connected component in $G[W]$ such that $0 < \alpha(W_\ell \cap W) < \alpha(C)$. 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_ℓ 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. ◀

Hence, we derive the following branching rule.

▶ **Branching Rule 5.2.** *Let $I = (G = (V, E), S, k, d)$ be an instance of GLS-VC such that there is no connected solution for I. Moreover, for each $j \in [1, \lfloor \frac{d}{2} \rfloor]$, let W_j denote a minimum valid connected j-improving $(k-d+j)$-swap for S in G. For each swap $W \in \big\{\{w\} \mid w \in N(W_j), j \in [1, \lfloor \frac{d}{2} \rfloor]\big\} \cup \big\{W_j \mid j \in [1, \lfloor \frac{d}{2} \rfloor]\big\}$, branch into the case of swapping W.*

With these branching rules, we are now able to prove Theorem 5.2. To obtain the stated running time of Theorem 5.2, we apply a branching algorithm using three steps in each node of the branching tree. 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 5.1. Due to Proposition 5.3, this is correct. If there is no vertex $v \in S$ with $N(v) \subseteq S$, find some minimum valid connected j-improving k-swaps W_j for I where $|W_j \setminus S| \leq (k-d)/2$ (if such a swap exists), for each $j \in [1, d]$. Due to Proposition 5.4, 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 5.2.

Note that this 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 [18] to GLS-VC.

5.2 Parameterizing Weighted Gap Local Search by Maximum Degree

▶ **Proposition 5.6 (*).** *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS-WVC. One can enumerate all valid connected k-swaps for I in $\mathcal{O}(2^k(\Delta-1)^k \cdot k^3 \cdot n)$ time.*

Due to Observation 3.4 and Proposition 5.6 we obtain the following.

▶ **Corollary 5.7.** *LS-WVC can be solved in $\mathcal{O}(2^k \cdot (\Delta-1)^k \cdot k^3 \cdot n)$ time.*

To solve GLS-WVC, again, we encounter the problem that the sought solution 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 which is, more or less, an adaptation of Branching Rule 5.2 to the weighted version. Consider a solution 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 5.3 cannot be obtained[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 $(\Delta-1)^{2\cdot k}$ instead of $(\Delta-1)^k$. Our goal is, thus, to reduce 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 solution 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 reduced by one. Hence, these ideas lead to the following branching rule.

▶ **Branching Rule 5.3.** *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS-WVC such that I has no connected solution. For each $j \in [1, \lfloor \frac{k}{2} \rfloor]$, let W_j denote some valid connected j-swap for I with maximal improvement. Branch into the case of swapping W for each swap $W \in \{\{w\} \mid w \in N(W_1)\} \cup \{W_j \mid j \in [1, \frac{k}{2}]\} \cup \{\{w\} \mid w \in N(W_j) \setminus S_1, j \in [2, \frac{k}{2}]\}$.*

With this branching rule, we are now able to show the following.

▶ **Theorem 5.8 (*).** *GLS-WVC can be solved in $\mathcal{O}(k! \cdot (\Delta-1)^k \cdot n)$ time.*

To obtain this running time, we do the following in each node of the branching tree. First, find for each $j \in [1, k]$ some valid connected j-swap W_j for I that maximizes $\alpha(W_j)$. Due to Proposition 5.6, this can be done in $\mathcal{O}(2^k \cdot (\Delta-1)^k \cdot k^3 \cdot n)$ time. Now, if $\alpha(W_k) \geq d$, then I is a yes-instance of GLS-WVC. Otherwise, there is no connected solution for I, since W_k has the maximal improvement of all valid connected k-swaps. Compute the set $S_1 := \{v \in S \mid N(v) \subseteq S\}$ of possible improving swaps of size 1 and apply Branching Rule 5.3.

Finally, we show that we can replace $\Delta(G)$ in the above running time by the h-index of 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, we compute the corresponding swap instance and solve it with the help of Theorem 5.8 after removing the remaining high-degree vertices. This is correct due to the following lemma.

Since a valid swap W for S in G that avoids a given set V' does not contain any vertex of $S \setminus V'$ adjacent to some vertex of $V' \setminus S$, we define 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 $S' := S \cap V(G')$. Due to the above, we obtain the following.

▶ **Lemma 5.9 (*).** *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS-WVC and let $V' \subseteq V$. There is a solution W for I with $W \cap V' = \emptyset$ if and only if the exclusion instance I' of V' and I is a yes-instance of GLS-WVC.*

▶ **Theorem 5.10 (*).** *GLS-WVC can be solved in $\mathcal{O}(k! \cdot (h-1)^k \cdot n)$ time.*

[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.

6 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.

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 6.1 (*).** *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 [22]. Note that $\mathcal{T}$ has $\mathcal{O}(n)$ quotient graphs. 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 $\alpha_S(W)$ 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 ℓ 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_ℓ 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 ith vertex of S_x according to some arbitrary but fixed ordering with $i \in [1, |S_x|]$. Moreover, let $V_x^{\geq i} = \bigcup_{j=i}^{|S_x|} V_{S_x(j)}$. The dynamic programming table $Q_{S_x}[i, k']$ has entries for $i \in [1, |S_x|+1]$ and $k' \in [0, k]$ and stores the maximal improvement of a valid k'-swap W for $S \cap V_x^{\geq i}$ in $G[V_x^{\geq i}]$, such that there is at least one vertex in $S \cap W \cap V_{S_x(j)}$ for each $j \in [i, |S_x|]$. We set

$$Q_{S_x}[i, k'] := \begin{cases} -\infty & |S_x| - i + 1 > k', \\ 0 & i = |S_x| + 1, \text{and} \\ \max_{1 \leq k'' \leq k'} D_{S_x(i)}[k''] + Q_{S_x}[i+1, k'-k''] & \text{otherwise.} \end{cases}$$

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 $W^* := \bigcup_{y \in N_x(S_x)} (V_y \setminus S)$.

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.

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. For a set of size x, let $\binom{x}{\leq k}$ 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 (the proof of this fact is deferred to Appendix B).

Since the value of $Q_{S_x}[1, k']$ is only evaluated one time 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. ◀

With a slight modification, we can improve the running time for GLS-VC.

▶ **Corollary 6.2.** *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.*

We also obtain an FPT-algorithm for GLS-VC for a new parameter that is upper-bounded by the maximum degree $\Delta(G)$ and by the modular-width $\mathrm{mw}(G)$. We call this parameter the *maximum modular degree* and it is defined by taking the maximum degree over all quotient graphs of a modular decomposition of minimum width.

▶ **Definition 6.3.** *Let* $(\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*), \beta)$ *be a modular decomposition of a graph* G*. Then the* maximum modular degree *of* $(\mathcal{T}, \beta)$ *is* $\Delta_{md}(\mathcal{T}, \beta) := \max_{x \in \mathcal{V}} \Delta(\beta(x))$*. Moreover, the* maximum modular degree $\Delta_{md}(G)$ *of* G *is the maximum modular degree of a modular decomposition* $(\mathcal{T}', \beta')$ *of* G *that minimizes* $\Delta_{md}(\mathcal{T}', \beta')$*.*

Since $\mathrm{mw}(G)$ is the largest vertex count of any quotient graph, we have $\Delta_{md}(G) < \mathrm{mw}(G)$. Moreover, the graph $\beta(x)$ is isomorphic to an induced subgraph of G for all x, and thus $\Delta_{md}(G) \leq \Delta(G)$.

▶ **Proposition 6.4 (*).** *Let* $(\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*), \beta)$ *be a modular decomposition of a graph* G *of minimum width where the quotient graph* $\beta(x)$ *is prime for each* $x \in \mathcal{V}$*. Then,* $\Delta_{md}(\mathcal{T}, \beta) = \Delta_{md}(G)$*.*

▶ **Theorem 6.5 (*).** *GLS-VC can be solved in* $\left(\Delta_{\mathrm{md}}(G)(k^2 + 2)\right)^k \cdot n^{\mathcal{O}(1)}$ *time.*

In the algorithm for $\Delta_{\mathrm{md}}(G)$, to avoid considering all k-swaps of a quotient graph $\beta(x)$, we instead adapt the algorithm of Section 5.2 to find suitable swaps of $\beta(x)$. The main difficulty is that we need to consider all possibilities of how many vertices have been swapped in the subtree of $\beta(x)$.

7 Conclusion

We introduced the notion of FPT running times that grow mildly with respect to some parameter ℓ and strongly with respect to another parameter k. Such running times are desirable in the setting where the parameter k is much smaller than ℓ. Parameterized local search is one scenario in which this assumption is certainly true, when k is the operational parameter that bounds the size of the local search neighborhood. We showed that such running times are achievable for one of the most important graph problems in parameterized local search, LS-Vertex Cover, and different well-known structural parameters taking the place of ℓ.

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. This could also be relevant in other scenarios with operational parameters, for example in turbo-charging of greedy algorithms [2, 8, 12]. Second, it is open to improve our running time bounds for LS-Vertex Cover 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. Furthermore, in our study, we did not explicitly consider permissive local search, where one may report better solutions outside of the local neighborhood if they exist [13]. Our positive and negative results also work in this setting, but it would be interesting to identify structural parameters ℓ 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 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? Moreover, can the performance of the parameterized local search algorithm for unweighted Vertex Cover with parameter (Δ, k) [18] be improved by some of the techniques presented in this work?

References

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. doi:10.1137/16M1061771.

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.

3 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA '21)*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.

4 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

5 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.

6 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.

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

8 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.

9 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.

10 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.

11 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. doi:10.1145/3186898.

12 Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julián Mestre, and Stefan Rümmele. Turbocharging treewidth heuristics. *Algorithmica*, 81(2):439–475, 2019. doi:10.1007/s00453-018-0499-1.

13 Serge Gaspers, Eun Jung Kim, Sebastian Ordyniak, Saket Saurabh, and Stefan Szeider. Don't be strict in local search! In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI '12)*. AAAI Press, 2012.

14 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.

15 Jiong Guo, Danny Hermelin, and Christian Komusiewicz. Local search for string problems: Brute-force is essentially optimal. *Theoretical Computer Science*, 525:30–41, 2014.

16 Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

17 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.

18 Maximilian Katzmann and Christian Komusiewicz. Systematic exploration of larger local search neighborhoods for the minimum vertex cover problem. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, (AAAI '17)*, pages 846–852. AAAI Press, 2017.

19 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.

20 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.

21 Dániel Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Operations Research Letters*, 36(1):31–36, 2008.

22 Ross M. McConnell and Jeremy P. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 536–545. ACM/SIAM, 1994.

23 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

24 Stefan Szeider. The parameterized complexity of k-flip local search for SAT and MAX SAT. *Discrete Optimization*, 8(1):139–145, 2011.

25 Gerhard J. Woeginger. Space and time complexity of exact algorithms: Some open problems (invited talk). In *Proceedings of the First International Workshop on Parameterized and Exact Computation (IWPEC '04)*, volume 3162 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 2004.

A Tree Decompositions and Treewidth

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 $\beta : \mathcal{V} \to 2^V$ such that

1. for every vertex $v \in V$, there is at least one $x \in \mathcal{V}$ with $v \in \beta(x)$,
2. for each edge $\{u, v\} \in E$, there is at least one $x \in X$ such that $u \in \beta(x)$ and $v \in \beta(x)$, 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 reachable from x in $\mathcal{T}$. Moreover, we set $G_x := G[V_x]$ and $E_x := E_G(V_x)$.

B Bounding the Number of Small Subsets

To obtain small polynomial factors in the running times of our algorithms, we show the following.

▶ **Lemma B.1 (*).** *Let $k \geq 1$ be an integer and let X be an arbitrary set of size $x \geq 3$. Moreover, let $\binom{x}{\leq k}$ denote the number of different subsets of X of size at most k. 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 $\binom{x}{\leq k} \leq x^k$. If $k \leq 4$, then $x^k \leq 16 \cdot (x-1)^k$ and since $k^2 \leq 16$, $\binom{x}{\leq k} \leq 256 \cdot (x-1)^k / k^2$. 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 $k \geq x/2$, $\binom{x}{\leq k} \leq 2^x \leq 4^k \leq 8^k / 2^k \leq 8^k / k^2 \leq (x-1)^k / k^2$. 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 $k \geq 2$. If $4 < k < x/2$, then $2 \cdot \binom{x}{k} \geq \sum_{r=0}^{k} \binom{x}{r} = \binom{x}{\leq k}$. Hence $\binom{x}{\leq k} \leq 2 \cdot \binom{x}{k} \leq 2 \cdot x! / (x-k)! \cdot 1/k! \leq 2 \cdot x \cdot (x-1)! / (x-k)! \cdot 1/k^2 \leq 2 \cdot 2(x-1) \cdot (x-1)^{k-1} \cdot 1/k^2 = 4(x-1)^k / k^2 < 256 \cdot (x-1)^k / k^2$. ◀