Fantastic Flips and Where to Find Them: A General Framework for Parameterized Local Search on Partitioning Problems

Niels Grüttemeier
Fraunhofer IOSB-INA, Lemgo, Germany

Nils Morawietz
Institute of Computer Science, Friedrich Schiller University Jena, Germany
LaBRI, Université de Bordeaux, Talence, France

Frank Sommer
Institute of Logic and Computation, TU Wien, Austria

Abstract

Parameterized local search combines classic local search heuristics with the paradigm of parameterized algorithmics. While most local search algorithms aim to improve given solutions by performing one single operation on a given solution, the parameterized approach aims to improve a solution by performing k simultaneous operations. Herein, k is a parameter called *search radius* for which the value can be chosen by a user. One major goal in the field of parameterized local search is to outline the trade-off between the size of k and the running time of the local search step.

In this work, we introduce an abstract framework that generalizes natural parameterized local search approaches for a large class of partitioning problems: Given n items that are partitioned into b bins and a target function that evaluates the quality of the current partition, one asks whether it is possible to improve the solution by removing up to k items from their current bins and reassigning them to other bins. Among others, our framework applies for the local search versions of problems like Cluster Editing, Vector Bin Packing, and Nash Social Welfare. Motivated by a real-world application of the problem Vector Bin Packing, we introduce a parameter called *number of types* $\tau \leq n$ and show that all problems fitting in our framework can be solved in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time, where $|I|$ denotes the total input size. In case of Cluster Editing, the parameter τ generalizes the well-known parameter neighborhood diversity of the input graph.

We complement these algorithms by showing that for all considered problems, an algorithm significantly improving over our algorithm with running time $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ would contradict the Exponential Time Hypothesis. Additionally, we show that even on very restricted instances, all considered problems are W[1]-hard when parameterized by the search radius k alone. In case of the local search version of Vector Bin Packing, we provide an even stronger W[1]-hardness result.

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

Keywords and phrases Flip-Neighborhood, Cluster Editing, Vector Bin Packing, Vertex Cover, NP-hard problem, Max c-Cut

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.32

Related Version *Continuously Updated Version*: `https://arxiv.org/abs/2506.24001`

Funding *Niels Grüttemeier*: Supported by the project *Datenfabrik.NRW*, a project by *KI.NRW*, funded by the Ministry for Economics, Innovation, Digitalization and Energy of the State of North Rhine-Westphalia (MWIDE).
Nils Morawietz: Partially supported by the French ANR, project ANR-22-CE48-0001 (TEMPOGRAL).
Frank Sommer: Supported by the Alexander von Humboldt Foundation.

Acknowledgements We would like to thank Lea Hoeing, Kai Lebbing, Christiane Markuse-Schneider, and Lukas Ptock (Schmitz Cargobull AG) for showing us the interesting application of Vector Bin Packing.

1 Introduction

The principle of local search is among the most important heuristic approaches in combinatorial optimization and it is highly relevant to find good solutions to NP-hard problems in practice [19]. The idea is to apply small modifications on a given starting solution to obtain a new solution with a better target value than the starting solution. Local search has been studied extensively and it has been proven to be highly efficient [2, 19, 26, 1]. Furthermore, it is easy to understand and it is also a good plugin to improve already competitive solutions provided by other metaheuristics [31, 10].

Consider a classic partitioning problem as MULTI KNAPSACK, where the goal is to assign items (each with a weight) to multiple knapsacks (each with a weight capacity and specific values for the items) in a way that all capacity constraints are satisfied and the total value is maximal. One of the most natural ways to apply small modifications in a local search scenario is an *item flip*, where one removes a single item from its current knapsack and inserts it into another knapsack. This natural idea of flipping the assignment of single items to improve a solution is studied for knapsack problems [4] and for other partitioning problems [8, 27]. A general drawback in performing these single item flips – and also in local search in general – is the chance of getting stuck in poor local optimal solutions. Thus, to obtain a robust local search application, a strategy to prevent getting stuck in these poor local optima is required.

In this work, we consider the approach of *parameterized local search* [29, 7] to decrease the chance of getting stuck in poor local optima reached by item flips. Instead of searching for an improving solution by *one* single operation (here: a single item flip), the user may set a search radius k to extend the search space for possible improvements that can be reached with up to k simultaneous operations. Thus, the idea is to make the search space larger so that getting stuck in poor local optima becomes less likely. Parameterized local search combines local search with the paradigm of parameterized algorithmics [3] and aims to outline the trade-off between the size of the search radius k and the running time of the search step. Parameterized local search has been studied extensively in the algorithmic community: for vertex deletion and partitioning problem in graphs [7, 13, 16, 18, 5, 21, 11, 10], for problems on strings and phylogenetic networks [17, 23], and many other problems [29, 32, 9, 15]. One major goal in parameterized local search is to show that finding an improvement within search radius k is fixed parameter tractable (FPT) for k. That is, finding an improving solution can be done in time $g(k) \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the total encoding length of the input instance and g is some computational function only depending on k. Note that an algorithm with such a running time nicely outlines the trade-off between radius size and running time, as the superpolynomial part only depends on k while $|I|$ only contributes as a polynomial factor to the running time. Unfortunately, most parameterized local search problems are W[1]-hard for k [29, 7, 17, 30] and therefore, an algorithm with running time $g(k) \cdot |I|^{\mathcal{O}(1)}$ presumably does not exist.

Motivated by the negative results for the parameter k, one often studies the combination of k and some structural parameter τ to obtain algorithms with running $g(k, \tau) \cdot |I|^{\mathcal{O}(1)}$. This approach has been successful both from a theoretical and experimental perspective [18, 21, 12, 10]. In this work, we follow this direction by studying parameterization by k and an additional parameter τ that we call the *number of types*. We consider the local search versions of a large class of well-known combinatorial problems including MAX c-CUT, MULTI KNAPSACK, CLUSTER EDITING and VECTOR BIN PACKING, where the search space is defined by performing k flips. Recall that one flip intuitively removes one item from its assigned set and inserts in into some other set. The interpretation of flips and of our parameter τ always depends on the concrete problem and will be explained for each problem individually.

Our approach of exploiting the parameter τ is motivated by a real-world production planning application at the company Schmitz Cargobull AG. Planning the production at Schmitz Cargobull AG corresponds to solving an instance of Vector Bin Packing [28]. In Vector Bin Packing, one is given a large collection of vectors $\mathcal{S} \subseteq \mathbb{N}^d$ together with a vector $w \in \mathbb{N}^d$ and an integer b. The question is, whether there exists a partition of $\mathcal{S}$ into b parts $S_1, \dots, S_b$, such that $\sum_{v \in S_i} v \leq w$ for all $i \in [1, b]$. In the application at Schmitz Cargobull AG, we have $\mathcal{S} \subseteq \{0,1\}^d$ and the vectors $v \in \mathcal{S}$ correspond to customer orders where the entries of v specify whether a specific option is chosen ($\widehat{=}$ the entry has value 1) or not ($\widehat{=}$ the entry has value 0). The entries of w correspond to the production capacities available for the corresponding product option at one day of production. Instead of asking for a partition where each resulting vector set satisfies $\sum_{v \in S_i} v \leq w$, one asks for a partition minimizing the *total overload*. More precisely, the *overload* of a single set S_i is defined as $\sum_{j=1}^{d} \max(0, (\sum_{v \in S_i} v_j) - w_j)$, and the *total overload* is the sum of all overloads of all sets $S_1, \dots, S_b$. Note that the total overload is 0 if and only if the given instance is a yes-instance of the decision version of Vector Bin Packing. While Schmitz Cargobull AG usually receives a large number of customer orders, relatively many of these orders request the exact same product option combinations. Consequently, the number of distinct vectors in $\mathcal{S}$ is much smaller than $|\mathcal{S}|$. Therefore, our research is motivated by setting τ to be the number of distinct vectors in an instance of Vector Bin Packing and study parameterized local search for the combination of k and τ where the target is to minimize the total overload.

From a more abstract point of view, Vector Bin Packing is a problem where one assigns a collection of *items* (vectors) to a collection of *bins* (sets of the resulting partition) in a way that a target function (total overload) is minimized. Furthermore, if two elements from $\mathcal{S}$ have the same vector, these elements have the same effect on the target function. In other words, there are only τ distinct ways in which a specific item might have an influence on the target function when assigned to a specific bin. This more abstract view leads to a framework providing running times $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ and $k^{\mathcal{O}(\tau)} \cdot |I|^{\mathcal{O}(1)}$ for the parameterized local search versions for a wide range of well-known combinatorial problems that behave in the same way as Vector Bin Packing. The $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ running-time is particularly motivated since the value of k is a small constant chosen by the user [24].

Recall that the concrete interpretation of the parameter τ, of the items, and of the bins always depends on the concrete problem and will be explained for each problem individually. Besides the number of distinct vectors, τ can be interpreted as the neighborhood diversity of an input graph in case of Max c-Cut or Cluster Editing, or as the number of distinct value-weight combinations in case of Multi Knapsack.

Our Contributions. In the first part (see Section 3), we introduce the LS Generalized Bin Problem, which is a general framework capturing many parameterized local search problems where the search radius is defined by a number of item flips. Moreover, we introduce the parameter *number of types* τ as an abstract concept for LS Generalized Bin Problem capturing well-known parameters such as the neighborhood diversity of a graph. We describe general algorithms for the abstract LS Generalized Bin Problem leading to running times of $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ and $k^{\mathcal{O}(\tau)} \cdot |I|^{\mathcal{O}(1)}$ for the local search versions of many problems that fit into our framework (see Theorem 3.5).

In the second part (see Section 4) we provide simple example applications of the introduced framework leading to new results for the parameterized local search versions of classic combinatorial optimization problems, graph problems, and one problem from computational social choice (Nash Social Welfare). The formal problem definitions can be found in

Table 1 The corresponding local search problems for the flip distance of these problems can be solved in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time and in $k^{\mathcal{O}(\tau)} \cdot |I|^{\mathcal{O}(1)}$ time for the respective parameter τ as we show in Section 4. In case of VERTEX COVER, we have exactly two bins: the resulting vertex cover (vc) and the remaining independent set (is). In this case, the flip distance corresponds to the cardinality of the symmetric difference between the current vertex cover and the improving vertex cover.

Problem	Items	Bins	Parameter τ	Section
MAX c-CUT	vertices	color classes	neighborhood diversity	4.1
CLUSTER EDITING	vertices	clusters	neighborhood diversity	4.2
VECTOR BIN PACKING	vectors	bins	number of distinct vectors	4.3
VERTEX COVER	vertices	(vc, is)	neighborhood diversity	4.4
NASH SOCIAL WELFARE	items	agents	number of distinct items	4.5
MULTI KNAPSACK	items	knapsacks	number of distinct items	4.5

Section 4. All results in this part mainly rely on simply reformulating the concrete problem as LS GENERALIZED BIN PROBLEM. An overview of the studied problems is given in Table 1. To the best of our knowledge, this is the first work studying parameterized local search for NASH SOCIAL WELFARE, VECTOR BIN PACKING, and MULTI KNAPSACK.

In Section 5, we complement our results by studying parameterization by the search radius k alone. We show that the parameterized local search versions of NASH SOCIAL WELFARE and MULTI KNAPSACK are W[1]-hard when parameterized by k. Furthermore, we provide a strong hardness result for VECTOR BIN PACKING showing W[1]-hardness for $k + q$, where q denotes the maximum number of non-zero-entries over all vectors from the input. The parameter q is particularly motivated by the real-world application from the company Schmitz Cargobull AG, where q is even smaller than τ. We finally show that all of our algorithms with running time $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ are tight in the sense of the *Exponential Time Hypothesis (ETH)* [20].

2 Preliminaries

For details on parameterized complexity, we refer to the standard monographs [3, 6].

For integers a and b with $a \leq b$, we define $[a, b] := \{i \in \mathbb{N} \mid a \leq i \leq b\}$. Given a set X and some integer b, we call a mapping $f : X \to [1, b]$ a *b-partition of X* or a *b-coloring of X*. For every $i \in [1, b]$, we let $f^{-1}(i) := \{x \in X \mid f(x) = i\}$. The *flip* between two b-partitions f and f' is defined as $D_{\text{flip}}(f, f') := \{x \in X \mid f(x) \neq f'(x)\}$. The *flip distance* between f and f' is then defined as $d_{\text{flip}}(f, f') := |D_{\text{flip}}(f, f')|$. For a set X, we let 2^X denote the *power set* of X.

For a graph $G = (V, E)$, by $n := |V|$ we denote the *number of vertices* and by $m := |E|$ we denote the *number of edges*. By $N(u) := \{w \in V \mid \{u, w\} \in E\}$ and by $N(S) := (\bigcup_{u \in S} N(u)) \setminus S$ we denote the *open neighborhood* of u and S, respectively. Two vertices u and w have the *same neighborhood class* if $N(u) \setminus \{w\} = N(w) \setminus \{u\}$. By $\text{nd}(G)$ we denote the number of neighborhood classes of G. A vertex set S is a *vertex cover* for G if each edge of E has at least one endpoint in S. A vertex set S is an *independent set* of G, if no edge has of E has both endpoints in S.

Proofs of statements marked with ($\star$) are deferred to the full version.

3 Generalized Parameterized Local Search for Partitioning Problems

We present a framework that captures many computational problems such as Vector Bin Packing, Cluster Editing, and Nash Social Welfare. On a high level, all these problems have in common, that one aims to partition some set X (e.g. a set of vectors, a set of vertices, or a set of items) into multiple "bins". A target function assigns a value to the resulting partition. The goal is to find a partition that minimizes (or maximizes) the target function. This section is structured as follows. We first introduce a "Generalized Bin Problem" that generalizes the computational problems studied in this work. Afterwards, we introduce a parameter called "types". Finally, we provide the algorithmic results for this parameter.

3.1 The Generalized Bin Problem

Intuitively, we aim to partition a given set X into a given number of "bins" $b \in \mathbb{N}$, and a target value specifies how good this b-partition of X is. This target value is determined by an "individual bin evaluation", which is a collection of local target values of each single bin. These values are then combined to obtain the target value for the whole b-partition of X.

▶ **Definition 3.1.** *Let X be a set, let $b \in \mathbb{N}$, and let* $\inf \in \{\infty, -\infty\}$. *An* individual bin evaluation (IBE) *is a b-tuple $(\varphi_i)_{i\in[1,b]}$ of functions $\varphi_i : 2^X \to \mathbb{Z} \cup \{\inf\}$. An IBE defines a* target value $\mathrm{val}(f)$ *for every b-partition f by*

$$\mathrm{val}(f) := \left(\bigoplus_{i=1}^{b} \varphi_i(f^{-1}(i)) \right),$$

where $\oplus$ is a commutative and associative binary operation $\oplus : \mathbb{Z}\cup\{\inf\}\times\mathbb{Z}\cup\{\inf\} \to \mathbb{Z}\cup\{\inf\}$ satisfying $a \oplus \inf = \inf \oplus\, a = \inf$ for all $a \in \mathbb{Z} \cup \{\inf\}$.

While our general framework works for arbitrary commutative and associative operations $\oplus$, this work only considers concrete problems where $\oplus$ is either the summation or the multiplication of integer numbers. The value $\inf \in \{\infty, -\infty\}$ corresponds to infeasible assignments of bins, for example, violating capacity constraints of a knapsack. In case of a minimization problem we consider IBE with $\inf = \infty$ and in case of a maximization problem we have $\inf = -\infty$. In the remainder of this section, all problem statements and algorithms are given for the case where one aims to minimize the target function. Maximization problems are defined analogously. With Definition 3.1 at hand, we define the following general computational problem for every fixed commutative and associative operation $\oplus$.

Generalized Bin Problem
Input: An integer b, a set X, and an IBE $(\varphi_i)_{i\in[1,b]}$.
Goal: Find a b-partition f that minimizes $\mathrm{val}(f)$.

Note that we did not yet specify how the IBE from the input is given. As this depends on the concrete problems, this will be discussed for each problem individually. Throughout this section, we analyze the algorithms running times with respect to the parameter Φ denoting the running time needed for one evaluation of a value $\varphi_i(X')$ with $X' \subseteq X$.

Recall that our aim is to study parameterized local search for the flip neighborhood. More precisely, we aim to find a b-partition f' that has a better target value than some given b-partition f, while $d_{\text{flip}}(f, f') \le k$ for a given k. The corresponding computational problem is defined as follows.

LS Generalized Bin Problem
Input: An integer b, a set X, and an IBE $(\varphi_i)_{i\in[1,b]}$, a b-partition $f : X \to [1, b]$, and an integer k.
Question: Is there a b-partition f' with $d_{\text{flip}}(f, f') \leq k$ such that $\text{val}(f') < \text{val}(f)$?

3.2 Types in Generalized Bins

We next define a parameter called "number of types" τ. The idea is, that in a polynomial-time preprocessing step, the set X is partitioned into classes of elements $(X_1, \dots, X_\tau)$ in a way that all elements in each X_i have the exact same impact on the target value for every possible bin assignment. The intuitive idea of "same impact" is formalized as follows.

▶ **Definition 3.2.** *Let X be a set, let $b \in \mathbb{N}$, and let $(\varphi_i)_{i\in[1,b]}$ be an IBE for X and b. Two (not necessarily distinct) elements $x \in X$ and $y \in X$ are* target equivalent *($x \sim y$), if for every $i \in [1, b]$ and for every $A \subseteq X$ with $\{x, y\}\cap A = \{x\}$ we have $\varphi_i((A\setminus\{x\})\cup\{y\}) = \varphi_i(A)$.*

▶ **Proposition 3.3** ($\star$)**.** *The relation $\sim$ is an equivalence relation on X.*

The main idea of our algorithm is that target equivalent elements can be treated equally as they have the same influence on the target value. When considering concrete problems, we always use a simple pairwise relation between the elements leading to a partition of X into classes of pairwise target equivalent elements. These classes do not necessarily need to be maximal under this constraint. Thus, it suffices to consider the following relaxation of the equivalence classes.

▶ **Definition 3.4.** *Let X be a set, let $b \in \mathbb{N}$, and let $(\varphi_i)_{i\in[1,b]}$ be an IBE for X and b. A tuple $(X_1, \dots, X_\tau)$ of disjoint sets with $\bigcup_{j=1}^{\tau} X_j = X$ is called a* type partition *of X if the elements of each X_j are pairwise target equivalent. For every $j \in [1, \tau]$, we say that the elements of X_j* have type *j.*

Obviously, the equivalence classes for $\sim$ always form a type partition with the minimum number of sets. Throughout this work, we assume that each instance $I := (b, X, (\varphi_i)_i, f, k)$ of LS Generalized Bin Problem is associated with a specified type partition, and the parameter *number of types* $\tau := \tau(I)$ is defined as the number of sets of the type partition associated with I.

3.3 Algorithmic Results for LS Generalized Bin Problem

Our goal is to study LS Generalized Bin Problem parameterized by $k+\tau$. Applying this on concrete problems then leads to FPT algorithms for a great range of parameterized local search versions of well-known computational problems. The interpretation of the parameter τ always depends on the concrete problem. We provide the following algorithmic results. Recall that Φ denotes the running time needed for one evaluation of a value $\varphi_i(X')$ with $X' \subseteq X$.

▶ **Theorem 3.5.** *LS Generalized Bin Problem can be solved*
a) *in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$ time, and*
b) *in $k^{\mathcal{O}(\tau)} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$ time*
if a type partition $(X_1, \dots, X_\tau)$ is additionally given as part of the input.

To present the algorithms behind Theorem 3.5, we introduce the notion of *type specification* and *type specification operations.* Recall that elements of the same type behave in the same way when assigned to a bin. Thus, when evaluating the target function, one may consider the types of the assigned elements instead of the concrete elements. A type specification is a vector $\vec{p}$ that intuitively corresponds to a collection of elements in X containing exactly p_j elements from the class X_j.

▶ **Definition 3.6.** *Let b be an integer, let X be a set, and let $(\varphi_i)_{i\in[1,b]}$ be an IBE. Moreover, let $(X_1, \dots, X_\tau)$ be a type partition. A* type specification *is a vector $\vec{p} = (p_1, \dots, p_\tau) \in \mathbb{N}_0{}^\tau$ with $p_j \in [0, |X_j|]$ for each $j \in [1, \tau]$.*

Since elements of the same type have the same impact on the target function, we can address the change of target values $\varphi_i(X')$ by just specifying the types of elements added to and removed from X' with $X' \subseteq X$. To ensure that after adding and removing elements of specific types from a set X' corresponds to an actual subset of X, we introduce the notion of subtractive and additive compatibility.

▶ **Definition 3.7.** *Let b be an integer, let X be a set, and let $(\varphi_i)_{i\in[1,b]}$ be an IBE. Moreover, let $(X_1, \dots, X_\tau)$ be a type partition. We say that a type specification $\vec{p}$ is*

a) subtractive compatible *with a set $X' \subseteq X$ if for every $j \in [1, \tau]$, we have $|X_j \cap X'| \geq p_j$.*
b) additive compatible *with a set $X' \subseteq X$ if for every $j \in [1, \tau]$, we have $|X_j \cap (X \setminus X')| \geq p_j$.*

We use $(\vec{p}, \vec{q}) \propto X'$ to denote that $\vec{p}$ is subtractive compatible with X' and $\vec{q}$ is additive compatible with X'. Given $\vec{p}$ and $\vec{q}$ with $(\vec{p}, \vec{q}) \propto X'$, the type vector operation *for each $i \in [1, b]$ is defined as*

$$\varphi_i((X' \setminus \vec{p}) \cup \vec{q}) := \varphi_i((X' \setminus X_{\vec{p}}) \cup X_{\vec{q}}),$$

where $X_{\vec{p}} \subseteq X$ is some set containing exactly p_j arbitrary elements from $X_j \cap X'$ for every $j \in [1, \tau]$, and $X_{\vec{q}} \subseteq X$ is some set containing exactly q_j arbitrary elements from $X_j \cap (X \setminus X')$ for every $j \in [1, \tau]$.

The notion of target equivalence (Definition 3.2) together with the notion of subtractive and additive compatibility guarantees the following.

▶ **Proposition 3.8.** *The type vector operation is well-defined.*

Proof. Since $\vec{p}$ is subtractive compatible with X', there are at least p_j elements in $X_j \cap X'$ and thus, the set $X_{\vec{p}} \subseteq X$ exists. Analogously, the set $X_{\vec{q}}$ exists due to the additive compatibility of $\vec{q}$ with X'. Consequently, $(X' \setminus X_{\vec{p}}) \cup X_{\vec{q}} \subseteq X$ and therefore, $\varphi_i((X' \setminus X_{\vec{p}}) \cup X_{\vec{q}})$ is defined.

It remains to show that the value $\varphi_i((X' \setminus X_{\vec{p}}) \cup X_{\vec{q}})$ does not depend on the choice of elements in $X_{\vec{p}}$ and $X_{\vec{q}}$. First, let $x \in X_{\vec{p}}$ and let $y \notin X_{\vec{p}}$ with $y \sim x$. Then,

$$\varphi_i(X' \setminus (X_{\vec{p}} \setminus \{x\} \cup \{y\}) \cup X_{\vec{q}}) = \varphi_i((X' \setminus X_{\vec{p}} \cup X_{\vec{q}}) \setminus \{y\} \cup \{x\}) = \varphi_i(X' \setminus X_{\vec{p}} \cup X_{\vec{q}}),$$

by Definition 3.2. Analogously, let $x \in X_{\vec{q}}$ and let $y \notin X_{\vec{q}}$ with $x \sim y$ we have

$$\varphi_i(X' \setminus X_{\vec{p}} \cup (X_{\vec{q}} \setminus \{x\} \cup \{y\})) = \varphi_i((X' \setminus X_{\vec{p}} \cup X_{\vec{q}}) \setminus \{x\} \cup \{y\}) = \varphi_i(X' \setminus X_{\vec{p}} \cup X_{\vec{q}}).$$

Therefore, the type vector operation is well-defined. ◀

With the notion of type specifications at hand, we next present the algorithmic results. Intuitively, the algorithms behind Theorem 3.5 work as follows: One considers every possibility of how many elements of which type belong to the (at most) k elements in $D_{\text{flip}}(f, f')$. For each such choice, one computes the best improving flip corresponding to the choice. Note that the information how many elements of which type are flipped, can be encoded as a type specification $\vec{\delta}$ in the way that all δ_j correspond to the number of flipped elements of type j.

We first describe the subroutine computing the improving b-partition f' corresponding to a given type specification $\vec{\delta}$. Afterwards, we describe how to efficiently iterate over all possible $\vec{\delta}$ to obtain the running times stated in Theorem 3.5.

We use the notation $\vec{p} \leq \vec{q}$ to indicate that $p_i \leq q_i$ for all vector entries, and we let $\vec{p} - \vec{q}$ denote the component-wise difference between $\vec{p}$ and $\vec{q}$.

▶ **Lemma 3.9.** *Let b be an integer, let X be a set, let $(\varphi_i)_{i\in[1,b]}$ be an IBE, and let $f : X \to [1,b]$ be a b-partition. Moreover, let $(X_1,\dots,X_\tau)$ be a type partition and let $\vec{\delta}$ be a type specification and we let $k := \sum_{j=1}^{\tau} \delta_j$. A b-partition $f' : X \to [1,b]$ with*

$$(|X_1 \cap D_{\text{flip}}(f,f')|,\dots,|X_\tau \cap D_{\text{flip}}(f,f')|) \leq \vec{\delta}$$

that minimizes $\text{val}(f')$ among all such b-partitions can be computed in
a) $2^{\mathcal{O}(k)} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$ *time, or in*
b) $(\lceil \frac{k}{\tau} \rceil + 1)^{4\tau} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$ *time.*

Proof. We first provide an algorithm based on dynamic programming and afterwards we discuss the running times *a*) and *b*).

Intuition. Before we formally describe the dynamic programming algorithm, we provide some intuition: Every $x \in D_{\text{flip}}(f,f')$ gets removed from its bin and is inserted into a new bin. We fix an arbitrary ordering of the bins and compute solutions for prefixes of this ordering in a bottom-up manner. For every bin, there is a set of removed elements R and a set of inserted elements I. Since the target value depends on the types of the elements rather than the concrete sets R and I, we may only consider how many elements of which type are removed from a bin and inserted into a bin. Therefore, we compute the partial solutions for given $\vec{p} \leq \vec{\delta}$ and $\vec{q} \leq \vec{\delta}$, where $\vec{p}$ corresponds to the removed types, and $\vec{q}$ corresponds to the inserted types. After the computation, we consider the solution, where $\vec{p} = \vec{q} = \vec{\delta}$, that is, the solution where the exact same types were removed and inserted. Going back from abstract types to concrete elements then yields the resulting b-partition f'.

Dynamic programming algorithm. The dynamic programming table has entries of the form $T[\vec{p},\vec{q},\ell]$ where $\ell \in [1,b]$, $\vec{q} \leq \vec{\delta}$, and $\vec{p} \leq \vec{\delta}$. One such table entry corresponds to the minimal value of

$$\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus R_i) \cup I_i)$$

under every choice of sets $R_i \subseteq f^{-1}(i)$ and $I_i \subseteq X \setminus f^{-1}(i)$ for $i \in [1,\ell]$ that satisfy

$$(\sum_{i=1}^{\ell} |R_i \cap X_1|,\dots,\sum_{i=1}^{\ell} |R_i \cap X_\tau|) = \vec{p} \quad \text{and} \quad (\sum_{i=1}^{\ell} |I_i \cap X_1|,\dots,\sum_{i=1}^{\ell} |I_i \cap X_\tau|) = \vec{q}.$$

The table is filled for increasing values of ℓ. As base case, we set $T[\vec{p},\vec{q},1] := \varphi_1((f^{-1}(1)\setminus\vec{p})\cup\vec{q})$ if $(\vec{p},\vec{q}) \propto f^{-1}(1)$. Recall that $(\vec{p'},\vec{q'}) \propto f^{-1}(1)$ is true if and only if $\vec{p'}$ is subtractive compatible with $f^{-1}(1)$ and $\vec{q'}$ is additive compatible with $f^{-1}(1)$. If $\vec{p}$ or $\vec{q}$ is incompatible, we set $T[\vec{p},\vec{q},1] := \infty$. The recurrence to compute an entry with $\ell > 1$ is

$$T[\vec{p},\vec{q},\ell] := \min_{\substack{\vec{p'} \leq \vec{p},\ \vec{q'} \leq \vec{q} \\ (\vec{p'},\vec{q'}) \propto f^{-1}(\ell)}} T[\vec{p}-\vec{p'},\vec{q}-\vec{q'},\ell-1] \oplus \varphi_\ell((f^{-1}(\ell) \setminus \vec{p'}) \cup \vec{q'}),$$

Note that $(\vec{0},\vec{0}) \propto f^{-1}(\ell)$ is always true, and therefore, such a minimum always exists. Herein, $\vec{0}$ denotes the τ-dimensional vector where each entry is 0.

▷ Claim 3.10. The recurrence is correct.

Proof. Throughout the proof of this claim, we call the properties posed on the sets $(R_1, \dots, R_\ell)$ and $(I_1, \dots, I_\ell)$ in the definition of the table entries the *desired properties for $\vec{p}$, $\vec{q}$, and ℓ.*

We show that the recurrence is correct via induction over ℓ. If $\ell = 1$, we have $T[\vec{p}, \vec{q}, 1] := \varphi_1((f^{-1}(1) \setminus \vec{p}) \cup \vec{q})$. Thus, by Definition 3.7, there are sets $R \subseteq f^{-1}(1)$ and $I \subseteq X \setminus f^{-1}(1)$ where R is a set containing exactly p_j elements from $X_j \cap f^{-1}(1)$ for every $j \in [1, \tau]$, and I is a set containing exactly q_j elements from $X_j \cap (X \setminus f^{-1}(1))$ for every $j \in [1, \tau]$ and we have $T[\vec{p}, \vec{q}, 1] = \varphi_1((f^{-1}(1) \setminus R) \cup I)$. Since the value is invariant under the concrete choices of elements in R and I due to Proposition 3.8, the value $\varphi_1((f^{-1}(1) \setminus R) \cup I)$ is minimal among all such R and I. Thus, the Recurrence holds for the base case $\ell = 1$.

Next, let $\ell > 1$ and assume that the recurrence holds for $\ell - 1$. Let $(R_1, \dots, R_\ell)$ and $(I_1, \dots, I_\ell)$ be sets with the desired properties for $\vec{p}$, $\vec{q}$, and ℓ. We show

$$\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus R_i) \cup I_i) = T[\vec{p}, \vec{q}, \ell].$$

($\geq$) Since $\oplus$ is associative and commutative, we have

$$\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus R_i) \cup I_i) = \underbrace{\left(\bigoplus_{i=1}^{\ell-1} \varphi_i((f^{-1}(i) \setminus R_i) \cup I_i)\right)}_{=:Z} \oplus \varphi_\ell((f^{-1}(\ell) \setminus R_\ell) \cup I_\ell).$$

Note that the vectors $\vec{a} := (|R_\ell \cap X_1|, \dots, |R_\ell \cap X_\tau|)$ and $\vec{b} := (|I_\ell \cap X_1|, \dots, |I_\ell \cap X_\tau|)$ satisfy $\vec{a} \leq \vec{p}$ and $\vec{b} \leq \vec{q}$. Moreover, since $R_\ell \subseteq f^{-1}(\ell)$ and $I_\ell \subseteq X \setminus f^{-1}(\ell)$, it holds that $(\vec{a}, \vec{b}) \propto f^{-1}(\ell)$ is true. Thus, the minimum of the right-hand-side of the recurrence includes $\vec{a}$ and $\vec{b}$, and by the definition of type specification operations and Proposition 3.8, we have $\varphi(\ell)((f^{-1}(\ell) \setminus \vec{a}) \cup \vec{q}) = \varphi(\ell)((f^{-1}(\ell) \setminus R_\ell) \cup I_\ell)$. Since by the inductive hypothesis we have $T[\vec{p} - \vec{a}, \vec{q} - \vec{b}, \ell - 1] \leq Z$, we conclude $\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus R_i) \cup I_i) \geq T[\vec{p}, \vec{q}, \ell]$.

($\leq$) Let $\vec{a}$ and $\vec{b}$ be the vectors minimizing the right-hand-side of the recurrence. By the definition of type specification operations, there are sets $\widetilde{R_\ell} \subseteq f^{-1}(\ell)$ and $\widetilde{I_\ell} \subseteq X \setminus f^{-1}(\ell)$ with $(|\widetilde{R_\ell} \cap X_1|, \dots, |\widetilde{R_\ell} \cap X_\tau|) = \vec{a}$ and $(|\widetilde{I_\ell} \cap X_1|, \dots, |\widetilde{I_\ell} \cap X_\tau|) = \vec{b}$. By inductive hypothesis, the value $T[\vec{p} - \vec{a}, \vec{q} - \vec{b}, \ell - 1]$ corresponds to the minimal value of $\bigoplus_{i=1}^{\ell-1} \varphi_i((f^{-1}(i) \setminus \widetilde{R_i}) \cup \widetilde{I_i})$ for sets $(\widetilde{R_1}, \dots, \widetilde{R_{\ell-1}})$ and $(\widetilde{I_1}, \dots, \widetilde{I_{\ell-1}})$ that satisfy the desired properties for $\vec{p} - \vec{a}$, $\vec{q} - \vec{b}$, and $\ell - 1$. Then, $T[\vec{p}, \vec{q}, \ell]$ corresponds to the value $\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus \widetilde{R_i}) \cup \widetilde{I_i})$ for sets $(\widetilde{R_1}, \dots, \widetilde{R_\ell})$ and $(\widetilde{I_1}, \dots, \widetilde{I_\ell})$ that satisfy the desired properties for $\vec{p}$, $\vec{q}$, and ℓ. Thus, it follows that $\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus R_i \cup I_i)) \leq T[\vec{p}, \vec{q}, \ell]$. ◁

Computation of a solution f'. We next describe how to compute the desired b-partition f' after filling the dynamic programming table T. The sets $(R_1, \dots, R_b)$ and $(I_1, \dots, I_b)$ corresponding to the table entry $T[\vec{\delta}, \vec{\delta}, b]$ can be found via trace back. We let $\mathcal{R} := \bigcup_{i=1}^{b} R_i$. Due to the property $R_i \subseteq f^{-1}(i)$ for all $i \in [1, b]$, all R_i are disjoint. Together with the properties on the vectors $\vec{p}$ and $\vec{q}$, this implies $|\mathcal{R} \cap X_j| = \sum_{i=1}^{b} |R_i \cap X_j| = \sum_{i=1}^{b} |I_i \cap X_j| = \delta_j$ for every $j \in [1, \tau]$. Consequently, for every $j \in [1, \tau]$, there is a mapping $\gamma_j : \mathcal{R} \cap X_j \to [1, b]$, that maps exactly $|I_i \cap X_j|$ elements of $\mathcal{R} \cap X_j$ to i for every $i \in [1, b]$. Intuitively, the mapping γ_j assigns a new bin to each item of type j that was removed from some bin.

The b-partition f' is defined via the mappings γ_j as follows: For every $x \in X \setminus \mathcal{R}$, we set $f'(x) := f(x)$. For every $x \in \mathcal{R}$, we set $f'(x) := \gamma_j(x)$ for the corresponding j with $x \in \mathcal{R} \cap X_j$.

By the definition of f', the b-partitions f and f' may only differ on $\mathcal{R}$ and therefore, $|\mathcal{R} \cap X_j| = \delta_j$ implies $(|X_1 \cap D_{\text{flip}}(f, f')|, \dots, |X_\tau \cap D_{\text{flip}}(f, f')|) \leq \vec{\delta}$. It remains to show that $\text{val}(f')$ is minimal. By the construction of f' we have $f'^{-1}(i) = f^{-1}(i) \setminus R_i \cup I'_i$, where $I'_i := \bigcup_{j=1}^{\tau} \{x \in \mathcal{R} \cap X_j \mid \gamma_j(x) = i\}$. Then, by the construction of the γ_j we have

$$\begin{aligned}(|I'_i \cap X_1|, \dots, |I'_i \cap X_\tau|) &= (|\{x \in \mathcal{R} \cap X_1 \mid \gamma_1(x) = i\}|, \dots, |\{x \in \mathcal{R} \cap X_\tau \mid \gamma_\tau(x) = i\}|) \\ &= (|I_i \cap X_1|, \dots, |I_i \cap X_\tau|).\end{aligned}$$

Thus, the sets $(f^{-1}(i) \setminus R_i) \cup I'_i$ and $(f^{-1}(i) \setminus R_i) \cup I_i$ contain the exact same number of elements from each X_j. Consequently, we have $\varphi_i((f^{-1}(i) \setminus R_i) \cup I'_i) = \varphi((f^{-1}(i) \setminus R_i) \cup I_i)$ for every $i \in [1, b]$ and thus, by the definition of types, the minimality of $T[\vec{\delta}, \vec{\delta}, b]$ implies the minimality of $\text{val}(f')$.

Running time. We finally provide two different ways to analyze the running time.

a) The vector $\vec{\delta}$ can be regarded as a set M containing $\sum_{j=1}^{\tau} \delta_j = k$ individual identifiers from which exactly δ_j are labeled with type j for each $j \in [1, \tau]$. With this view on $\vec{\delta}$, the vectors $\vec{p} \leq \vec{\delta}$ and $\vec{q} \leq \vec{\delta}$ correspond to subsets of M, so we have 2^k possible choices for $\vec{p}$ and 2^k possible choices for $\vec{q}$. Consequently, the size of the table T is $2^{2k} \cdot b$. To compute one entry, one needs to consider up to 2^{2k} choices of $\vec{p'}$ and $\vec{q'}$. For each such choice, $(\vec{p'}, \vec{q'}) \propto f^{-1}(\ell)$ can be checked in $|X|^{\mathcal{O}(1)}$ time. With the evaluation of the IBE, this leads to a total running time of $2^{4k} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$.

b) The number of possible vectors that are component-wise smaller than $\vec{\delta}$ is $\prod_{j=1}^{\tau}(\delta_j + 1)$, since each entry is an element of $[0, \delta_j]$. Since $\sum_{j=1}^{\tau} \delta_j = k$, the product of all $(\delta_j + 1)$ is maximal if all δ_j have roughly the same size $\frac{k}{\tau}$. Thus, we have $\prod_{j=1}^{\tau}(\delta_j + 1) \leq (\frac{k}{\tau} + 1)^\tau$. Consequently, the size of T is upper-bounded by $(\lceil \frac{k}{\tau} \rceil + 1)^{2\tau} \cdot b$. To compute one entry, one needs to consider up to $(\lceil \frac{k}{\tau} \rceil + 1)^{2\tau}$ choices of $\vec{p'}$ and $\vec{q'}$. For each such choice, $(\vec{p'}, \vec{q'}) \propto f^{-1}(\ell)$ can be checked in $|X|^{\mathcal{O}(1)}$ time. With the evaluation of the IBE, this leads to a total running time of $(\lceil \frac{k}{\tau} \rceil + 1)^{4\tau} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$. ◀

We next use Lemma 3.9 to prove Theorem 3.5.

Proof of Theorem 3.5. Recall that the idea is to consider every possible vector $\vec{\delta}$ specifying how many elements of which type belong to the elements of the flip. For each possible $\vec{\delta}$ we then use the algorithm behind Lemma 3.9. It remains to describe how to efficiently iterate over the possible choices of $\vec{\delta}$ to obtain the running times a) and b).

a) Let $\vec{e_1}, \dots, \vec{e_\tau}$ be the τ-dimensional unit vectors. That is, only the jth entry of $\vec{e_j}$ equals 1 and all other entries equal 0. We enumerate all $\vec{\delta}$ with $\sum_{i=1}^{\tau} \delta_i \leq k$ by considering all τ^k possibilities to repetitively draw up to k-times from the set $\{\vec{e_1}, \dots, \vec{e_\tau}\}$. For each such choice $\vec{\delta}$, we check whether $\delta_i \leq |X_i|$ for each $i \in [1, \tau]$ to ensure that $\vec{\delta}$ is a type specification. If this is the case, we apply the algorithm behind Lemma 3.9. With the running time from Lemma 3.9 a), this leads to a total running time of $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$.

b) Note that for a vector $\vec{\delta}$ with $\sum_{i=1}^{\tau} \delta_i = k$, we have $\delta_i \in [0, k]$ for every $i \in [1, \tau]$. Thus, in time $(k + 1)^\tau \cdot k^{\mathcal{O}(1)}$ we can enumerate all possible $\vec{\delta}$. Analogously to a), we check whether $\delta_i \leq |X_i|$ for each $i \in [1, \tau]$. With the running time from Lemma 3.9 b), this leads to a total running time of $k^{\mathcal{O}(\tau)} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$. ◀

4 Applications of the Framework

We next study parameterized local search versions of a wide range of well-known problems. For each of these problems, we consider parameterization by the search radius in combination with some structural parameter. All results rely on the algorithm behind Theorem 3.5.

Therefore, it suffices to express an instance I of a local search problem as a corresponding instance $J = (b, X, (\varphi_i)_{i\in[1,b]}, f, k)$ of LS Generalized Bin Problem. The proofs in this section are given in the following structure:

1. *GBP Construction:* Given the instance I of the local search problem, we specify the universe X, the number of bins b and the IBE $(\varphi_i)_{i\in[1,b]}$ of the instance J. This also includes a description of how to efficiently evaluate the IBE from the input instance I.
2. *Type Partition:* We express how a type partition for J is computed from I and describe how the targeted structural parameter corresponds to the number of types as defined in Section 3.
3. *Solution Correspondence:* Recall that an instance of a local search problem always contains a solution of the underlying problem. Let s be the solution given in the instance I. To show the solution correspondence, we describe how solutions of I can be transformed into solutions of J and vice versa, such that the following holds: A solution s' of I is better than the given solution s of I if and only if the corresponding b-partition $f_{s'}$ has strictly smaller (larger) target value than f_s with respect to the IBE $(\varphi_i)_{i\in[1,b]}$.

4.1 Max c-Cut

Let $c \in \mathbb{N}$ and let $G = (V, E)$ be an undirected graph. We say that an edge $e \in E$ is properly colored by a c-partition χ of V, if χ assigns distinct colors to the endpoints of e.

Max c-Cut
Input: An undirected graph $G = (V, E)$.
Goal: Find a c-partition χ of V that maximizes the number of properly colored edges under χ.

Note that the goal of Max c-Cut can also be equivalently redefined as: Find a c-partition χ of V that minimizes the number of edges that are not properly colored under χ, that is, a c-partition χ that minimizes $\text{faults}(\chi) := \sum_{i\in[1,c]} |E_G(\chi^{-1}(i))|$.

The input of the corresponding local search problem LS Max c-Cut additionally consists of a c-partition χ and some $k \in \mathbb{N}$, and one aims to find a c-partition χ' with $\text{faults}(\chi') < \text{faults}(\chi)$ and $d_{\text{flip}}(\chi, \chi') \le k$.

LS Max c-Cut is W[1]-hard parameterized by k [10] and cannot be solved in $f(k) \cdot n^{o(k)}$ time unless the ETH is false [30]. Form a positive side, it was shown that the problem can be solved in $2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time on apex-minor-free graphs [7]. Moreover, on general graphs, the problem can be solved in $\Delta^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time [10], where Δ denotes the maximum degree of the input graph. This algorithms found successful application as a post-processing algorithm for a state-of-the-art heuristic for Max c-Cut [10].

We show the following.

▶ **Theorem 4.1.** *LS Max c-Cut can be solved in* $\text{nd}^k \cdot 2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ *and* $k^{\mathcal{O}(\text{nd})} \cdot n^{\mathcal{O}(1)}$ *time.*

Proof. The proof relies on the algorithm behind Theorem 3.5. We describe how to use this algorithm to solve an instance $I := (G = (V, E), \chi, k)$ of LS Max c-Cut.

1. GBP Construction. We describe how to obtain an instance $J := (b, X, (\varphi_i)_{i\in[1,b]}, f, k)$ of LS Generalized Bin Problem. Our universe X is exactly the vertex set V of G, and the number b of bins is exactly the number c of colors. We next define the IBE. For each $i \in [1, c]$ and each vertex set $S \subseteq V$, we define $\varphi_i(S) := |E_G(S)|$. Note that for each value $\varphi(S)$ can obviously be computed in $n^{\mathcal{O}(1)}$ time, and thus, the IBE can be evaluated in polynomial time from the input instance I. Our operation $\oplus$ is the sum of integer numbers.

2. Type Partition. Our type partition is the collection of neighborhood classes of G. Recall that this collection can be computed in $\mathcal{O}(n+m)$ time.

To show that this collection is in fact a type partition, we show that two vertices from the same neighborhood class are target equivalent according to Definition 3.2. To this end, let C be a neighborhood class of G, let u and v be vertices of C. We show that u and v are target equivalent. Let $S \subseteq V$ be a vertex set with $S \cap \{u,v\} = \{u\}$. We show that for each $i \in [1,b]$, $\varphi_i(S) = \varphi_i((S \setminus \{u\}) \cup \{v\})$. Let $S' := (S \setminus \{u\}) \cup \{v\}$. Since all IBEs $(\varphi_i)_{i\in[1,b]}$ are identically defined, it suffices to only consider $i = 1$. By the fact that C is a neighborhood class of G, u and v have the same neighbors in $S \setminus \{u\} = S' \setminus \{v\}$. Hence, $\varphi_1(S) = \varphi_1(S')$. This implies that u and v are target equivalent.

3. Solution Correspondence. Note that the set of solutions χ' of LS Max c-Cut is exactly the set of all c-partitions of V. Since $X = V$ and $b = c$, the solutions of I have a one-to-one correspondence to the solutions of J. Moreover, note that the definition of the IBE is based on the reformulation of the objective function of Max c-Cut. Since $\oplus$ is the sum of integer numbers, we have $\text{val}(\chi') = \text{faults}(\chi')$ for every c-partition χ'. Therefore, χ' is a better solution than χ for LS Max c-Cut if and only if χ' is a smaller target value than χ with respect to the IBE. ◀

4.2 Cluster Editing

A *cluster graph* is an undirected graph in which each connected component forms a clique. Let $G = (V,E)$ be an undirected graph. In Cluster Editing, one aims to apply a minimum number of edge modification (edge insertions and edge deletions) on G, such that the resulting graph is a cluster graph. We let $\binom{V}{2}$ denote the set of two-element subsets of vertices corresponding to edge modifications. Given a set $E' \subseteq \binom{V}{2}$, we let $E \triangle E' := (E \setminus E') \cup (E' \setminus E)$ denote the symmetric difference corresponding to the application of the graph modifications.

Cluster Editing
Input: An undirected graph $G = (V,E)$.
Goal: Find a set $E' \subseteq \binom{V}{2}$ such that $(V, E\triangle E')$ is a cluster graph and $|E'|$ is minimal under this property.

Note that the maximum number of clusters corresponds to the number of vertices n. We consider an equivalent definition of Cluster Editing where one asks for an n-labeling χ with partitioning V into n (possibly empty) classes $\chi^{-1}(1), \dots, \chi^{-1}(n)$, such that

$$\text{score}(\chi) := \sum_{i=1}^{n} |E(\chi^{-1}(i))| - (\binom{|\chi^{-1}(i)|}{2} - |E(\chi^{-1}(i))|)$$

is maximal [30]. Intuitively, maximizing this score corresponds to maximizing the number of present edges in the connected components minus the required edge insertions such that these components form a cliques.

We study a parameterized local search version of Cluster Editing where one flip in the flip neighborhood corresponds to moving single vertices from their current cluster into another cluster. The input of the corresponding local search problem LS Cluster Editing consists of the input of Cluster Editing, together with an additional integer k and an n-labeling χ of V for some. The task is to compute a n-labeling χ' such that $\text{score}(\chi') > \text{score}(\chi)$ and $d_{\text{flip}}(\chi, \chi') \leq k$.

With respect to this neighborhood, LS Cluster Editing is known to be W[1]-hard parameterized by k and cannot be solved in $f(k) \cdot n^{o(k)}$ time unless the ETH is false [11]. Form a positive side, it was shown that the problem can be solved in $\Delta^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time [11], where Δ denotes the maximum degree of the input graph.

We now show the following.

▶ **Theorem 4.2.** *LS Cluster Editing can be solved in* $\mathrm{nd}^k \cdot 2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ *time and in* $k^{\mathcal{O}(\mathrm{nd})} \cdot n^{\mathcal{O}(1)}$ *time.*

Proof. The proof relies on the algorithm behind Theorem 3.5. We describe how to use this algorithm to solve an instance $I := (G = (V, E), k, \chi)$.

1. GBP Construction. We describe how I corresponds to an LS Generalized Bin Problem-instance $J := (b, X, (\varphi_i)_{i \in [1,b]}, f, k)$. Our universe X is exactly the vertex set V of G, and the number b of bins is exactly n. We next define the IBE. For each $i \in [1, n]$ and each vertex set $S \subseteq V$, we define $\varphi_i(S) := |E(S)| - (\binom{|S|}{2} - |E(S)|)$. Note that each value $\varphi(S)$ can obviously be computed in $n^{\mathcal{O}(1)}$ time from the input graph G. Our operation $\oplus$ is the sum of integer numbers.

2. Type Partition. Our type partition is the collection of neighborhood classes of G. Recall that this collection can be computed in $\mathcal{O}(n + m)$ time.

To show that this collection is in fact a type partition, we show that two vertices from the same neighborhood class of G are target equivalent according to Definition 3.2. To this end, let C be a neighborhood class of G and let u and v be vertices of c. Let $S \subseteq V$ be a vertex set with $S \cap \{u, v\} = \{u\}$. We obviously have $|S| = |(S \setminus \{u\}) \cup \{v\}|$. Moreover, since u and v have the exact same neighbors in G, we have $|E(S)| = |E((S \setminus \{u\}) \cup \{v\})|$. Then, by the definition of the IBE we have $\varphi_i(S) = \varphi_i((S \setminus \{u\}) \cup \{v\})$ for all $i \in [1, n]$.

3. Solution Correspondence. Note that the set of solutions χ' of LS Cluster Editing is exactly the set of all n-partitions of V. Since $X = V$ and $b = n$, the solutions of J have a one-to-one correspondence to the solutions of J. Moreover, note that the definition of the IBE is based on the reformulation of the objective function of Cluster Editing and $\oplus$ is the sum of integer numbers, we have $\mathrm{val}(\chi') = \mathrm{score}(\chi')$ for every n-partition χ'. Therefore, χ' is a better solution than χ for Cluster Editing if and only if χ' has a larger target value than χ with respect to the IBE. ◀

4.3 Vector Bin Packing

Let b, d be integers. In this section for $i \in [1, b]$, B_i is a *bin*. Each bin B_i is associated with a *weight vector* $\omega_i \in \mathbb{N}^d$. Let $\mathcal{S}$ be a set of vectors from $\mathbb{N}_0^d$. A b-partition χ of $\mathcal{S}$ is called a *bin assignment*.

Given a subset $S \subseteq \mathcal{S}$, we say that the *overload of bin B_i with respect to S* is $\mathsf{ol}(B_i, S) := \sum_{j=1}^{d} \max\left(0, (\sum_{v \in S} v_j) - w_j\right)$. Our target is to find an assignment χ minimizing the total overload, which is defined as $\sum_{i \in [1,b]} \mathsf{ol}(B_i, \chi^{-1}(i))$.

Vector Bin Packing
Input: Integers b, d, a set $\mathcal{S}$ of vectors from $\mathbb{N}_0^d$, a vector $\omega_i \in \mathbb{N}_0^d$ for each $i \in [1, b]$.
Goal: Find a bin assignment χ of $\mathcal{S}$ such that the total overload is minimized, that is, $\sum_{i \in [1,b]} \mathsf{ol}(B_i, \chi^{-1}(i))$ is minimized.

The input of the corresponding local search problem LS Vector Bin Packing additionally consists of a bin assignment χ and some $k \in \mathbb{N}$, and one aims to find a bin assignment χ' with $\sum_{i \in b} \mathsf{ol}(B_i, \chi'^{-1}(i)) < \sum_{i \in b} \mathsf{ol}(B_i, \chi^{-1}(i)))$ and $d_{\mathrm{flip}}(\chi, \chi') \leq k$.

We say that two vectors vec_1 and vec_2 are *identical* if they agree in all dimensions, that is, if $\mathsf{vec}_1(j) = \mathsf{vec}_2(j)$ for each dimensions $j \in [1, d]$ and two vectors are *distinct* otherwise.

To the best of our knowledge, parameterized local search for Vector Bin Packing has not yet been studied. We now show the following.

▶ **Theorem 4.3.** *Let τ denote the maximum number of pairwise distinct items in an instance of LS Vector Bin Packing. LS Vector Bin Packing can be solved in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot (b + d + |\mathcal{S}|)^{\mathcal{O}(1)}$ time and in $k^{\mathcal{O}(\tau)} \cdot (b + d + |\mathcal{S}|)^{\mathcal{O}(1)}$ time.*

Proof. The proof relies on the algorithm behind Theorem 3.5. We describe how to use this algorithm to solve an instance

$$I := (b, d, \mathcal{S}, (\omega_i)_{i\in[1,b]}, \chi, k)$$

of LS Vector Bin Packing

1. GBP construction. Initially, we describe how I corresponds to an instance $J := (b, X, (\varphi_i)_{i\in[1,b]}, f, k)$ of LS Generalized Bin Problem. Our universe X is exactly the set $\mathcal{S}$ and the number b is the number of bins in I. It remains to define the IBE. For $i \in [1, b]$, we define $\varphi_i(X') := \mathsf{ol}(B_i, X')$. That is, the ith IBE corresponds to the overload of B_i with respect to the vectors X'. Our operation $\oplus$ is the sum of integer numbers.

2. Type Partition. Let $(X_1, \dots, X_\tau)$ be a partition of $\mathcal{S}$ where each X_j is an inclusion-wise maximal set of pairwise identical elements. Note tat this partition can clearly be computed in $|\mathcal{S}|^{\mathcal{O}(1)}$ time.

To show that this collection is in fact a type partition, we show that two elements from one set X_j are target equivalent according to Definition 3.2. Let vec_1 and vec_2 be two vectors of the same set X_j. Let $S \subseteq \mathcal{S}$ be a set with $S \cap \{\mathsf{vec}_1, \mathsf{vec}_2\} = \{\mathsf{vec}_1\}$. Let $S' := (S \setminus \{\mathsf{vec}_1\}) \cup \{\mathsf{vec}_2\}$. We show that for each $i \in [1, b]$, $\varphi_i(S) = \varphi_i(S')$. Without loss of generality, we consider the IBE φ_i. Since C is a type class containing vec_1 and vec_2, we have $S \setminus \{\mathsf{vec}_1\} = S' \setminus \{\mathsf{vec}_2\}$. Consequently, $\varphi_i(S \cup \{\mathsf{vec}_2\}) = \varphi_i(S' \cup \{\mathsf{vec}_1\})$ and thus vec_1 and vec_2 are target equivalent.

3. Solution Correspondence. Since $\oplus$ is the sum of integer numbers and $\varphi_i(X') = \mathsf{ol}(B_i, X')$ for all $i \in [1, b]$, we have $\mathrm{val}(\chi) = \sum_{i\in b} \mathsf{ol}(B_i, \chi^{-1}(i))$ for every bin assignment χ. ◀

In Theorem 5.3 we complement the above algorithm by showing that LS Vector Bin Packing is W[1]-hard parameterized by k and cannot be solved in $f(k) \cdot n^{o(k)}$ time unless the ETH fails.

4.4 Vertex Deletion Distance to Specific Graph Properties

We next consider a general class of graph problems. Given a graph, one aims to delete a minimum number of vertices such that the remaining graph satisfies a specific property (for example: being bipartite). For any fixed graph-property Π that can be verified in polynomial time, we define the following problem.

Π Vertex Deletion
Input: A graph $G = (V, E)$.
Goal: Find a subset $S \subseteq V$ such that $G - S$ fulfills property Π and $|S|$ is minimal.

Parameterized local search was considered for Π Vertex Deletion for general graph properties Π [7] and in particular for the special case of Π being the family of all edgeless graphs, that is, for Vertex Cover [7, 13, 21, 24]. For the corresponding local search problems LS Π Vertex Deletion and LS Vertex Cover, the considered local neighborhood is the *k-swap neighborhood*, that is, the set of all solutions that have a symmetric difference with the current solution of size at most k. This neighborhood coincides with the k-flip neighborhood if a solution for Π Vertex Deletion and Vertex Cover is represented as a 2-coloring χ of the vertex set V, where $G[\chi^{-1}(1)]$ fulfills property Π and the goal is to minimize the number of vertices that receive color 2 under χ.

It was shown that LS Π Vertex Deletion is W[1]-hard when parameterized by k for all hereditary graph properties Π that contain all edgeless graphs but not all cliques or vice versa [7]. This holds in particular for the special case of LS Vertex Cover [7, 13, 24]. Furthermore, a closer inspection shows that LS Vertex Cover cannot be solved in $f(k) \cdot n^{o(k)}$ time unless the ETH fails [24, Theorem 3.7]. Since LS Vertex Cover is a special case of LS Π Vertex Deletion, we also obtain the same hardness results for the more general LS Π Vertex Deletion. From the positive side, LS Vertex Cover admits an FPT-algorithm for k, if the input graph has a bounded local treewidth. Moreover, algorithms for LS Vertex Cover are known that run in $f(k) \cdot \ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time, where ℓ is any of (i) the maximum degree Δ [21], (ii) the h-index of the input graph [24], or (iii) the treewidth of the input graph [24]. In particular, the performance of the algorithm combining k and Δ was shown to be very successful [21].

We show that our framework is applicable for an even more general problem LS Multi Component Π Deletion in which we aim to find a vertex set of minimal size to remove, so that the remaining vertices can be partitioned into c sets, that each fulfill property Π. Note that LS Π Vertex Deletion is the special case of $c = 1$.

▶ **Theorem 4.4** (⋆). *LS Multi Component Π Deletion can be solved in* $\mathrm{nd}^k \cdot 2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ *time and in* $k^{\mathcal{O}(\mathrm{nd})} \cdot n^{\mathcal{O}(1)}$ *time.*

4.5 Further Applications of the Framework

With similar applications the algorithm behind Theorem 3.5, we can obtain results for the local search versions of the well-known problems Nash Social Welfare and Multi Knapsack. Here, we only present the problem definitions and the respective results. References to the related literature and the proofs of our theorems for both problems are deferred to the full version.

Nash Social Welfare. Let n, m be integers. In this section, $\mathcal{A}$ is a set of n agents and $\mathcal{S}$ is a set of m items. Furthermore, we have n utility functions $(u_i : \mathcal{S} \to \mathbb{N})_{i \in [1,n]}$. An n-partition χ of $\mathcal{S}$ is called an *allocation*. For an allocation χ, the function $\mathrm{Nash}(\chi) := \prod_{i \in [1,n]} \left(\sum_{s \in \chi^{-1}(i)} u_i(s) \right)$ is called the *Nash score* of χ.

Nash Social Welfare
Input: A set $\mathcal{A}$ of agents, a set $\mathcal{S}$ of items, and a set $(u_i)_{i \in [1,n]}$ of utilities.
Goal: Find an allocation χ that maximizes $\mathrm{Nash}(\chi)$.

The input of the corresponding local search problem LS Nash Social Welfare additionally consists of an allocation χ and some $k \in \mathbb{N}$ and one asks for an allocation χ' with $\mathrm{Nash}(\chi') > \mathrm{Nash}(\chi)$ and $d_{\text{flip}}(\chi, \chi') \leq k$.

We say that two items j_1 and j_2 are *identical* if each agents values j_1 and j_2 similarly, that is, if $u_i(j_1) = u_i(j_2)$ for each $i \in [n]$. Two items are *distinct* if they are not identical.

▶ **Theorem 4.5** ($\star$). *Let τ denote the maximum number of pairwise distinct items in an instance of LS Nash Social Welfare. LS Nash Social Welfare can be solved in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot (n+m)^{\mathcal{O}(1)}$ time and in $k^{\mathcal{O}(\tau)} \cdot (n+m)^{\mathcal{O}(1)}$ time.*

Multi Knapsack. Let $m \in \mathbb{N}$ be a number of knapsacks, with their capacities $W_1, \dots, W_m \in \mathbb{N}$ and let $[1, n]$ be a set of items with values $v_{(\ell,i)} \in \mathbb{N}$ and weights $w_{(\ell,i)} \in \mathbb{N}$ for all $\ell \in [1, n]$ and $i \in [1, m]$. We say that an $(m + 1)$-partition χ of $[1, n]$ *fits into the knapsacks*, if $\sum_{i \in \chi^{-1}(\ell)} w_{(\ell,i)} \leq W_i$ for every $i \in [1, m]$. We define the *score of* χ as $\text{score}(\chi) := \sum_{i=1}^{m} \sum_{\ell \in \chi^{-1}(i)} v_{(\ell,i)}$. Intuitively, for $i \in [1, m]$, the set $\chi^{-1}(i)$ corresponds to the chosen items for the ith knapsack and $\chi^{-1}(m + 1)$ corresponds to the set of non-chosen items.

Multi Knapsack
Input: Knapsacks with capacities $W_1, \dots, W_m \in \mathbb{N}$, items $[1, n]$ with values $v_{(\ell,i)} \in \mathbb{N}$ and weights $w_{(\ell,i)} \in \mathbb{N}$ for all $\ell \in [1, n]$ and $i \in [1, m]$.
Goal: Find an $(m+1)$-partition χ of $[1, n]$ that fits into the knapsacks, such that $\text{score}(\chi)$ is maximal.

The input of the corresponding local search problem LS Multi Knapsack additionally consists of an m-partition χ that fits into the knapsacks and some $k \in \mathbb{N}$ and one asks for an improving $(m + 1)$-partition χ' with $d_{\text{flip}}(\chi, \chi') \leq k$ that fits into the knapsacks.

We say that two items $\ell \in [1, n]$ and $\ell' \in [1, n]$ are *identical* if $v_{(\ell,i)} = v_{(\ell',i)}$ and $w_{(\ell,i)} = w_{(\ell',i)}$ for all $i \in [1, m]$. Two elements are distinct if they are not identical.

To the best of our knowledge, parameterized local search for Multi Knapsack has not yet been studied. We now show the following.

▶ **Theorem 4.6** ($\star$). *Let τ denote the maximum number of pairwise distinct items in an instance of LS Multi Knapsack. LS Multi Knapsack can be solved in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot (n + m)^{\mathcal{O}(1)}$ time and in $k^{\mathcal{O}(\tau)} \cdot (n+m)^{\mathcal{O}(1)}$ time.*

5 Intractability Results for the Considered Local Search Problems

In this section, we present (parameterized) intractability results for the considered local search problems in this work and tight running-time lower bounds with respect to the $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$-time algorithms derived in Section 4. As already discussed in the previous sections, some of the considered problems in this work are known to be W[1]-hard when parameterized by k and cannot be solved in $f(k) \cdot |I|^{o(k)}$ time, unless the ETH fails. This is the case for LS Max c-Cut [30], LS Cluster Editing [11], LS Π Vertex Deletion [7, 13, 24]. In this section we thus only show that these intractability results also hold for the remaining local search problems considered in this work, that is, for LS Multi Knapsack, LS Nash Social Welfare, and LS Vector Bin Packing.

We start by presenting our (parameterized) intractability results and running-time lower bounds for the local search problems LS Multi Knapsack and LS Nash Social Welfare.

▶ **Theorem 5.1** ($\star$). *LS Multi Knapsack is* W[1]*-hard with respect to k and cannot be solved in $n^{o(k)}$ time, unless the ETH fails, where n denotes the number of items in the input instance. This holds even if there is only a single knapsack.*

▶ **Theorem 5.2** (⋆). *LS Nash Social Welfare is* W[1]*-hard with respect to k and cannot be solved in $n^{o(k)}$ time, unless the ETH fails. This holds even if there are only two agents and both have the same evaluation for each item.*

Now, we provide matching hardness results for LS Vector Bin Packing. More precisely, we provide two hardness results even if each entry in each vector is only 0 or 1. First, we show that LS Vector Bin Packing is W[1]-hard with respect to k and that an algorithm with running time $f(k) \cdot |I|^{o(k)}$ violates the ETH. Consequently, our $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time algorithm presented in Theorem 4.3 is tight if the ETH is true. Second, we show W[1]-hardness for LS Vector Bin Packing parameterized by $k + q$. Here, $q := \max_{v \in \mathcal{S}} \|v\|_1$ is the maximal sum of entries over all vectors. Recall that q is usually smaller than τ in our real-world application.

▶ **Theorem 5.3.** *LS Vector Bin Packing is* W[1]*-hard with respect to k and cannot be solved in $f(k) \cdot |I|^{o(k)}$ time, unless the ETH fails. This holds even if $b = 2$, each entry in each vector is only 0 or 1, and each entry of the vector ω for each bin is 1.*

Proof. We present a simple reduction from LS Max c-Cut which provides the desired intractability results. Let $c \geq 2$ and let $I := (G = (V, E), \chi : V \to [1, c], k)$ be an instance of LS Max c-Cut where χ is locally optimal if and only if χ is globally optimal. Even under these restrictions, LS Max c-Cut is W[1]-hard with respect to k and cannot be solved in $f(k) \cdot n^{o(k)}$ time, unless the ETH fails [10]. Let $m := |E|$ and let $E := \{e_1, \dots, e_m\}$. We obtain an equivalent instance $I' := (c, m, \mathcal{S}, (\omega_i)_{i \in [1,c]}, \psi, k')$ of LS Vector Bin Packing as follows: For each bin $j \in [1, c]$, we define the vector ω_j as the vector of length m that has a 1 in each dimension. For each vertex $v \in V$, we create a vector x_v that has a 1 at dimension $i \in [1, m]$ if and only if vertex v is incident with edge e_i. Let $\mathcal{S}$ be the set of these vectors and let ψ be the coloring obtained from χ by assigning for each vertex $v \in V$, color $\chi(v)$ to vector x_v, that is, $\chi(v) = \psi(x_v)$. Finally, we set $k' := k$.

For the sake of simplicity, we may identify each vector x_v by its corresponding vertex v. Similarly, we may consider c-partitions of V instead of c-partitions of $\mathcal{S}$ as solutions for I' based on the obvious bijection between vertices and vectors. Next, we show that I is a yes-instance of LS Max c-Cut if and only if I' is a yes-instance of LS Vector Bin Packing. To this end, we first analyze the objective functions of both instances with respect to corresponding solutions.

Let χ' be a c-coloring of V and let ψ' be the corresponding c-coloring of $\mathcal{S}$. Let $e_i := \{u, v\}$ be an edge of G having endpoints of distinct color under χ'. Then, no bin produces an overload in dimension i because only the vectors x_u and x_v have a 1 at dimension i and both vectors receive distinct colors under ψ'. Similarly, let $e_i := \{u, v\}$ be an edge of G having endpoints of the same color $\alpha \in [1, c]$ under χ'. Then, no bin except α produces an overload at dimension i and bin α produces an overhead of exactly 1 in dimension i, because only the vectors x_u and x_v have a 1 at dimension i and both vectors receive color α under ψ'. Consequently, the total overload of ψ' over all dimensions and all bins equals $|E|$ minus the number of properly colored edges of G under χ'. In other words, χ' is a better solution for I than χ if and only if ψ' is a better solution for I' than ψ. Since $d_{\text{flip}}(\chi, \chi') = d_{\text{flip}}(\psi, \psi')$, this implies that I is a yes-instance of LS Max c-Cut if and only if I' is a yes-instance of LS Vector Bin Packing. Furthermore, since χ is a locally optimal solution if and only if χ is a globally optimal solution, ψ is a locally optimal solution if and only if ψ is a globally optimal solution.

Recall that LS Max c-Cut is W[1]-hard with respect to k and cannot be solved in $f(k) \cdot n^{o(k)}$ time, unless the ETH fails [10]. Since $|\mathcal{S}| = n$, each vector has $m \leq n^2$ dimensions, and $k' = k$, this implies that LS Vector Bin Packing is W[1]-hard with respect to k' and cannot be solved in $f(k') \cdot |I'|^{o(k')}$ time, unless the ETH fails. ◀

Now, we present our second hardness result for the parameter k plus q, the maximal sum of entries over all vectors.

▶ **Theorem 5.4** (⋆). *LS Vector Bin Packing is* W[1]*-hard with respect to $k + q$, even if each entry in each vector is only 0 or 1, and each entry of the vector ω for each bin is 1.*

6 Discussion

There are several ways of extending our work: In all of our applications of the framework we extensively used the expressive power of the IBEs and the flexibility of the bins. So far, we have not exploited the power of the types. For future work it is thus interesting to find examples where the number τ of types is smaller than the neighborhood diversity or the number of distinct vectors. Also, it is interesting to study the parameterized complexity of the non-local search versions of the considered problems parameterized by τ alone. For example, Bin Packing admits an FPT-algorithm when parameterized by τ alone [22]. It appears to be reasonable that the algorithm can be modified to obtain fixed-parameter tractability for Vector Bin Packing parameterized by τ as well. Furthermore, Vertex Cover admits an FPT-algorithm when parameterized by nd [25] and Max c-Cut admits an FPT-algorithm when parameterized by $\text{nd} + c$ [14]. However, it is not known whether such algorithms are possible for all (classic) variants of our study; for example for Nash Social Welfare such an algorithm is not known.

References

1 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. doi:10.4230/LIPICS.IPEC.2021.31.

2 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. doi:10.1613/JAIR.3907.

3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015. doi:10.1007/978-3-319-21275-3.

4 Juan A. Díaz and Elena Fernández. A tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research*, 132(1):22–38, 2001. doi:10.1016/S0377-2217(00)00108-9.

5 Martin Dörnfelder, Jiong Guo, Christian Komusiewicz, and Mathias Weller. On the parameterized complexity of consensus clustering. *Theoretical Computer Science*, 542:71–82, 2014. doi:10.1016/J.TCS.2014.05.002.

6 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

7 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. doi:10.1016/J.JCSS.2011.10.003.

8 Paola Festa, Panos M Pardalos, Mauricio GC Resende, and Celso C Ribeiro. Randomized heuristics for the max-cut problem. *Optimization methods and software*, 17(6):1033–1058, 2002. doi:10.1080/1055678021000090033.

9 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021. doi:10.1007/S00453-020-00758-8.

10 Jaroslav Garvardt, Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. Parameterized Local Search for Max c-Cut. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI '23)*, pages 5586–5594. ijcai.org, 2023. doi:10.24963/IJCAI.2023/620.

11 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. doi:10.4230/LIPICS.IPEC.2023.20.

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. doi:10.1609/AAAI.V26I1.8128.

14 Tomas Gavenciak, Martin Koutecký, and Dusan Knop. Integer programming in parameterized complexity: Five miniatures. *Discrete Optimization*, 44(Part):100596, 2022. doi:10.1016/J.DISOPT.2020.100596.

15 Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. Efficient Bayesian network structure learning via parameterized local search on topological orderings. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 12328–12335. AAAI Press, 2021. Full version available at https://doi.org/10.48550/arXiv.2204.02902. doi:10.1609/AAAI.V35I14.17463.

16 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. doi:10.1007/S00453-012-9685-8.

17 Jiong Guo, Danny Hermelin, and Christian Komusiewicz. Local search for string problems: Brute-force is essentially optimal. *Theoretical Computer Science*, 525:30–41, 2014. doi:10.1016/J.TCS.2013.05.006.

18 Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science*, 494:86–98, 2013. doi:10.1016/J.TCS.2012.12.049.

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

20 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

21 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. doi:10.1609/AAAI.V31I1.10659.

22 Dusan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. Parameterized complexity of configuration integer programs. *Operations Research Letters*, 49(6):908–913, 2021. doi:10.1016/J.ORL.2021.11.005.

23 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. doi:10.4230/LIPICS.CPM.2023.18.

24 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. doi:10.4230/LIPICS.IPEC.2022.20.

25 Martin Koutecký. Solving hard problems on neighborhood diversity. Master's thesis, Charles University in Prague, 2013. URL: https://koutecky.name/mgr/mgr.pdf.

26 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. doi:10.1080/01605682.2019.1621218.

27 Andrea Lodi, Silvano Martello, and Daniele Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112(1):158–166, 1999. doi:10.1016/S0377-2217(97)00388-3.

28 Christiane Markuse-Schneider. Personal communication. Schmitz Cargobull production site in Vreden, Germany, 2023.

29 Dániel Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008. doi:10.1016/J.ORL.2007.02.008.

30 Nils Morawietz. *On the complexity of local search problems with scalable neighborhoods*. PhD thesis, Friedrich-Schiller-Universität Jena, 2024. Dissertation. URL: https://www.db-thueringen.de/receive/dbt_mods_00064137.

31 Quan Ouyang and Hong Yun Xu. Genetic algorithm for single machine scheduling problem with setup times. *Applied Mechanics and Materials*, 457:1678–1681, 2014.

32 Stefan Szeider. The parameterized complexity of k-flip local search for SAT and MAX SAT. *Discrete Optimization*, 8(1):139–145, 2011. doi:10.1016/J.DISOPT.2010.07.003.