Solving Packing Problems with Few Small Items Using Rainbow Matchings

Max Bannach
Institute for Theoretical Computer Science, Universität zu Lübeck, Germany
bannach@tcs.uni-luebeck.de

Sebastian Berndt
Institute for IT Security, Universität zu Lübeck, Germany
s.berndt@uni-luebeck.de

Marten Maack
Department of Computer Science, Universität Kiel, Germany
mmaa@informatik.uni-kiel.de

Matthias Mnich
Institut für Algorithmen und Komplexität, TU Hamburg, Germany
mmnich@tuhh.de

Alexandra Lassota
Department of Computer Science, Universität Kiel, Germany
ala@informatik.uni-kiel.de

Malin Rau
Université Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, France
malin.rau@inria.fr

Malte Skambath
Department of Computer Science, Universität Kiel, Germany
malte.skambath@email.uni-kiel.de

Abstract

An important area of combinatorial optimization is the study of packing and covering problems, such as Bin Packing, Multiple Knapsack, and Bin Covering. Those problems have been studied extensively from the viewpoint of approximation algorithms, but their parameterized complexity has only been investigated barely. For problem instances containing no "small" items, classical matching algorithms yield optimal solutions in polynomial time. In this paper we approach them by their *distance from triviality*, measuring the problem complexity by the number k of small items.

Our main results are fixed-parameter algorithms for vector versions of Bin Packing, Multiple Knapsack, and Bin Covering parameterized by k. The algorithms are randomized with one-sided error and run in time $4^k \cdot k! \cdot n^{O(1)}$. To achieve this, we introduce a colored matching problem to which we reduce all these packing problems. The colored matching problem is natural in itself and we expect it to be useful for other applications. We also present a deterministic fixed-parameter algorithm for Bin Packing with run time $O((k!)^2 \cdot k \cdot 2^k \cdot n \log(n))$.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Bin Packing, Knapsack, matching, fixed-parameter tractable

Digital Object Identifier 10.4230/LIPIcs.MFCS.2020.11

Related Version A full version of the paper is available at `http://arxiv.org/abs/2007.02660`.

Funding *Matthias Mnich*: Supported by DFG grants MN 59/1-1 and MN 59/4-1.
Alexandra Lassota: Supported by DFG grant "Strukturaussagen und deren Anwendung in Scheduling- und Packungsprobleme", JA 612/20-1

Acknowledgements We want to thank Magnus Wahlström for helpful discussions.

1 Introduction

An important area of combinatorial optimization is the study of packing and covering problems. Central among those is the Bin Packing problem, which has sparked numerous important algorithmic techniques. In Bin Packing, the goal is to pack a set of n items with sizes in $(0, 1]$ into as few unit-sized bins as possible. Referring to its simplicity and vexing intractability, this problem has been labeled as "the problem that wouldn't go away" more than three decades ago [11] and is still the focus of groundbreaking research today. Regarding approximability, the best known is an additive $O(\log \mathrm{OPT})$-approximation, due to Hoberg and Rothvoß [13, 17].

A recent trend is to apply tools from parameterized complexity theory to problems from operations research [30]. For Bin Packing, a natural parameter is the minimum number of bins. For this parameter, Jansen et al. [19] showed that this problem is W[1]-hard, even for instances encoded in unary. Another natural parameter is the number d of distinct item sizes. For $d = 2$, a polynomial-time algorithm was discovered by McCormick et al. [28, 29] in the 1990s. The complexity for all $d \geq 3$ was open for more than 15 years, until a breakthrough result of Goemans and Rothvoß [13] showed that Bin Packing can be solved in time $(\log \Delta)^{2^{O(d)}}$, where Δ is the largest number in the input. A similar result was shown later by Jansen and Klein [18]. Neither the algorithm by Goemans and Rothvoß nor the algorithm by Jansen and Klein are fixed-parameter algorithms for parameter d, which would require the algorithm to run in time $f(d) \cdot n^{O(1)}$ for some computable function f[1].

In light of these daunting results, we propose another natural parameter for Bin Packing. This parameter is motivated by the classical approach of parameters measuring the *distance from triviality* – a concept that was first proposed by Niedermeier [32, Sect. 5.4]. Roughly speaking, this approach measures the distance of the given instance from an instance which is solvable in polynomial time. This approach was already used for many different problems such as Clique, Set Cover, Power Dominating Set, or Longest Common Subsequence [15]. Even one of the arguably most important graph parameters – treewidth – is often interpreted as the distance of a given graph from a tree [15]. Interestingly, the number of special cases where Bin Packing can be solved in polynomial time is rather small and the corresponding algorithms often rely on reductions to matching problems. In this work, we propose as novel parameter the distance from instances without *small items*. If no small item (with size at most 1/3) exists, Bin Packing becomes polynomial-time solvable via a reduction to the matching problem as each bin can contain at most two items. If the number of small items is unbounded, the problem becomes NP-hard.

Two related problems to Bin Packing are Bin Covering, where the number of covered bins (containing items of total size at least 1) should be maximized, and Multiple Knapsack – a generalization of the Knapsack problem. These problems have been studied extensively (see the books by Gonzalez [14] and Kellerer et al. [22]). They share the Bin Packing trait that the efficiency of exact algorithms is hindered by the existence of small objects.

In all mentioned problems, the items have a one-dimensional size requirement. As this is too restrictive in many applications, so-called *vector versions* were proposed [1, 10]. In these versions, called Vector Packing, Vector Covering, and Vector Multiple Knapsack, each object has a d-dimensional size requirement and a set of objects can be packed only if the size constraints are fulfilled in *each* dimension $j = 1, \ldots, d$. These problems are much harder than their 1-dimensional version, e.g., Vector Packing does not admit an asymptotic polynomial time approximation scheme even for $d = 2$ [38]. For d-dimensional problems, we use the word *vectors* instead of *items* and *containers* instead of *bins.*

[1] Although the algorithm by Jansen and Klein is a fixed-parameter algorithm for the parameter $|V_I|$ – the number of vertices of the integer hull of the underlying knapsack polytope.

What it means to be small. In the one-dimensional version of Vector Packing, the definition of a small item is quite natural: Every item with size less or equal than $1/3$ is considered small. As a consequence, each bin can contain at most two large items. We would like to transfer this property from one dimension to the d-dimensional case.

The requirement for large items is that only two of them can be placed inside the same container. We call a subset of vectors $\mathcal{V}' \subseteq \mathcal{V}$ *3-incompatible* if no selection of three distinct vectors from $\mathcal{V}'$ may be placed in the same container, i.e., for each $u, v, w \in \mathcal{V}'$ there exists an $\ell \in \{1, \dots, d\}$ such that $u^\ell + v^\ell + w^\ell > T^\ell$, where T^ℓ is the *capacity constraint* of the container in dimension ℓ. Let $\mathcal{V}_L \subseteq \mathcal{V}$ be a *largest* 3-incompatible set; we call the vectors $v \in \mathcal{V}_L$ *large* and call the vectors from the set $\mathcal{V}_S = \mathcal{V} \setminus \mathcal{V}_L$ *small.* Moreover, we define the *number of small vectors* in $\mathcal{V}$ as the cardinality of the complement of a largest 3-incompatible set in $\mathcal{V}$. Note that each 3-incompatible set $\mathcal{V}'$ contains at most two vectors where all the entries have size of at most $1/3$. Hence, for Bin Packing the largest 3-incompatible set corresponds to the set of large items plus at most two additional items.

An important property of our definition is that the smallness of a vector is no longer an attribute of the vector itself, but needs to be treated with regard to all other vectors. Finding a set $\mathcal{V}_S \subseteq \mathcal{V}$ of small vectors of minimum cardinality might be non-trivial. We argue that this task is fixed-parameter tractable parameterized by $|\mathcal{V}_S|$. To find $\mathcal{V}_S$, we compute the largest 3-incompatible set in $\mathcal{V}$. The complement $\mathcal{V}_S = \mathcal{V} \setminus \mathcal{V}_L$ of a largest 3-incompatible set can be found in time $f(|\mathcal{V}_S|) \cdot n^{O(1)}$ by a reduction to 3-Hitting Set. In this problem, a collection of sets $S_1, \dots, S_n \subseteq T$ with $|S_i| = 3$ is given, and a set $H \subseteq T$ with $H \cap S_i$ for all $i \in \{1, \dots, n\}$ is sought. In Section 3, we present a reduction from the problem of finding the sets $\mathcal{V}_L$ and $\mathcal{V}_S$ to an instance of the 3-Hitting Set problem, which we can solve using:

▶ **Fact 1** ([9, 33, 37]). 3-Hitting Set *can be solved in time $2.27^k \cdot n^{O(1)}$, where k is the size of the solution. A corresponding solution can obtained within the same time.*

Our results. We settle the parameterized complexity of the vector versions of Bin Packing, Bin Covering, and Multiple Knapsack parameterized by the number k of small objects. Our main results are randomized fixed-parameter algorithms, which solve all those problems in time $O(k!) \cdot n^{O(1)}$ with one-sided error where n is the total number of objects. Note that Vector Multiple Knapsack is already NP-hard for $d = 1$ and $p_{\max} \leq n^{O(1)}$ [12, 26] where $p_{\max}$ denotes the largest profit of any object.

▶ **Theorem 2.** Vector Packing *and* Vector Covering *can be solved by a randomized algorithm (with bounded false negative rate in n) in time $4^k \cdot k! \cdot n^{O(1)}$.* Vector Multiple Knapsack *can be solved by a randomized algorithm (with bounded false negative rate in $n + p_{\max}$) in time $4^k \cdot k! \cdot n^{O(1)} \cdot (p_{\max})^{O(1)}$ where $p_{\max}$ is the largest profit of any vector.*

Our approach is to reduce the vector versions of packing and covering problems to a new matching problem on edge-colored graphs, which we call Perfect Over-the-Rainbow Matching.

In the Perfect Over-the-Rainbow Matching problem, we are given a graph G. Each edge $e \in E(G)$ is assigned a set of colors $\lambda(e) \subseteq \mathcal{C}$ and for each color, there is a non-negative weight $\gamma(e, c)$. The objective is to find a perfect matching M of G and a function $\xi \colon M \to \mathcal{C}$ such that (i) $\chi(e) \in \lambda(e)$ for all $e \in M$ (we can only choose from the assigned colors), (ii) $\bigcup_{e \in M} \chi(e) = \mathcal{C}$ (every color is present in the matching), and (iii) $\sum_{e \in M} \gamma(e, \chi(e))$ is minimized (the sum of the weights is minimized). The parameter for the problem is $|\mathcal{C}|$, the number of different colors.

We show how to solve Perfect Over-the-Rainbow Matching by an approach that is based on the Conjoining Matching problem. The Conjoining Matching problem was proposed by Sorge et al. [36], who asked whether it is fixed-parameter tractable. The question was resolved independently by Gutin et al. [16], and by Marx and Pilipczuk [27], who both gave randomized fixed-parameter algorithms. Based on both results, we derive:

▶ **Theorem 3.** *There is a randomized algorithm (with bounded false negative rate in $n + \ell$) that solves* Perfect Over-The-Rainbow Matching *in time $2^{|C|} \cdot n^{O(1)} \cdot \ell^{O(1)}$.*

This algorithm forms the backbone of our algorithms for Vector Packing, Vector Covering, and Vector Multiple Knapsack.

Whether there is a deterministic fixed-parameter algorithm for Conjoining Matching remains a challenging question, as also pointed out by Marx and Pilipczuk [27]. For some of the problems that can be solved by the randomized algebraic techniques of Mulmeley, Vazirani and Vazirani [31], no deterministic polynomial-time algorithms have been found, despite significant efforts. The question is whether the use of such matching algorithms is essential for Conjoining Matching, or can be avoided by a different approach.

We succeed in circumventing the randomness of our algorithm in the 1-dimensional case of Bin Packing. Namely, we develop another, deterministic algorithm for Bin Packing, for which we prove strong structural properties of an optimal solution; those structural insights may be of independent interest.

▶ **Theorem 4.** Bin Packing *can be solved deterministically in time $O((k!)^2 \cdot k \cdot 2^k \cdot n \log(n))$.*

Due to space constraints, we relegate details and proofs to the full version of this paper available at `http://arxiv.org/abs/2007.02660`.

Related Work. The class of small items, their relation to matching problems, and special instances without small items have been extensively studied in the literature: Shor [34, 35] studies the relation between *online bin packing,* where the items are uniformly randomly chosen from $(0, 1]$, and the matching problem on planar graphs. Those problems are closely related as almost all bins in an optimal solution contain at most two items. Csirik et al. [5] study the Generalized First-Fit-Decreasing heuristic for Vector Packing and show that their strategy is optimal for instances that contain at most two small items. Kenyon [23] studies the expected performance ratio of the *Best-Fit* algorithm for Bin Packing on a worst-case instance where the items arrive in random order. To prove an upper bound on the performance ratio, she classifies items into small items (size at most $1/3$), medium items (size at least $1/3$ and at most $2/3$), and large items (size at least $2/3$) [23]. Kuipers [24] studies so-called *bin packing games* where the goal is to share a certain profit in a fair way between the players controlling the bins and players controlling the items. He only studies instances without small items and shows that every such instances has a non-empty ε-core (a way of spreading the profits relatively fair) for $\varepsilon \geq 1/7$. Babel et al. [2] present an algorithm with competitive ratio $1 + 1/\sqrt{5}$ for online bin packing without small items. In another online version of the problem, the items and a conflict graph on them are given offline and, then online, variable-sized bins arrive. The case that the conflict graph is the union of two cliques corresponds to instances with no small items and was studied by Epstein et al. [7]. Another version of Bin Packing forbids to pack more then k different items into a single bin. The special case $k = 2$ corresponds to instances without small items and can be solved in time $n^{O(1/\varepsilon^2)}$ for bins of size $1 + \varepsilon$ [8]. Finally, Bansal et al. [3] study approximation algorithms for Vector Packing. To obtain their algorithms, they present a structural lemma that states that any solution with m bins can be turned into a solution with $(d+1)m/2$ bins such that each bin either contains at most two items or has empty space left in all but one dimensions. This result is then used to reduce the problem to a multi-objective budgeted matching problem.

From an approximation point of view, the problems considered in this work have been studied extensively, both for the 1-dimensional variant as well as for the vector versions. We refer to the survey of Christensen et al. [4] for an overview. Regarding parameterized algorithms for problems from operations research, the resulting body of literature is too large for a detailed description and we refer to the survey of Mnich and van Bevern [30]. Some of the 1-dimensional variants of the problems considered in this work have been studied from a parameterized perspective [18, 19, 20]. In contrast, to the best of our knowledge, there are no such results for the vector versions of these problems.

2 Preliminaries: Parameterized and Randomized Algorithms

We give a short introduction to parameterized and randomized algorithms, and refer to the standard textbooks for details [6, 32]. Afterwards, we introduce the packing problems formally. Finally, we define our auxiliary matching problem.

Parameterized Algorithms. A *parameterized problem* is a language $L \subseteq \{0,1\}^* \times \mathbb{N}$ where the second element is called the *parameter*. Such a problem is *fixed-parameter tractable* if there is an algorithm that decides whether (x,k) is in L in time $f(k)\cdot|x|^c$ for a computable function f and constant c. A *parameterized reduction* from a parameterized problem L to another one L' is an algorithm that transforms an instance (x,k) into (x',k') such that (i) $(x,k) \in L \Leftrightarrow (x',k') \in L'$, (ii) $k' \leq f(k)$, and (iii) runs in time $f(k)\cdot|x|^c$.

Randomized Algorithms. A *randomized algorithm* is an algorithm that explores some of its computational paths only with a certain probability. A randomized algorithm A for a decision problem L has *one-sided error* if it either correctly detects positive or negative instances with probability 1. It has a *bounded false negative rate* if $\Pr[\mathsf{A}(x) = \text{“no”} \mid x \in L] \leq 1/|x|^c$, that is, it declares a “yes”-instance as a “no”-instance with probability at most $1/|x|^c$. All randomized algorithms in this article have bounded false negative rate.

Packing and Covering Problems. In the Vector Packing problem we aim to pack a set $\mathcal{V} = \{v_1,\dots,v_n\} \subseteq \mathbb{Q}^d_{\geq 0}$ of vectors into the smallest possible number of containers, where all containers have a common capacity constraint $T \in \mathbb{Q}^d_{\geq 0}$. Let $v_j \in \mathcal{V}$ be a vector. We use v_j^ℓ to denote the ℓ^{th} component of v_j and T^ℓ to denote the ℓ^{th} constraint. A *packing* is a mapping $\sigma\colon \mathcal{V} \to \mathbb{N}_{>0}$ from vectors to containers. It is *feasible* if all containers $i \in \mathbb{N}_{>0}$ meet the capacity constraint, that is, for each $\ell \in \{1,\dots,d\}$ it holds that $\sum_{v_j\in\sigma^{-1}(i)} v_j^\ell \leq T^\ell$. Using as few containers as possible means to minimize $\max\{\sigma(v_j) \mid v_j \in \mathcal{V}\}$.

In the introduction we already discussed what it means to be “small”. We expect only few small items, so we consider this quantity as parameter for Vector Packing:

Vector Packing *Parameter:* Number k of small vectors

Input: A set $\mathcal{V} = \{v_1,\dots,v_n\} \subseteq \mathbb{Q}^d_{\geq 0}$ vectors and capacity constraints $T \in \mathbb{Q}^d_{\geq 0}$.

Task: Find a packing of $\mathcal{V}$ into the smallest number of containers.

The 1-dimensional case of the problem is the Bin Packing problem. There, vectors are called *items*, their single component *size* and the containers *bins*. In contrast to the multi-dimensional case, we are now given a *sequence* of items, denoted as $\mathcal{I}$.[2]

Another related problem is Vector Covering, where we aim to *cover* the containers. We say a packing $\sigma\colon \mathcal{V} \to \mathbb{N}_{>0}$ covers a container i if $\sum_{v_j \in \sigma^{-1}(i)} v_j^\ell \geq T^\ell$ for each component $\ell \in \{1, \dots, d\}$. The objective is to find a packing σ that maximizes the number of covered containers, that is, we want to maximize $|\{i \in \mathbb{N} \mid \sum_{v \in \sigma^{-1}(i)} v_j^\ell \geq T^\ell \text{ for all } \ell \in \{1, \dots, d\}\}|$. The last problem we study is the Vector Multiple Knapsack problem: Here a packing into a finite number of C many containers is sought. Therefore, not all vectors may fit into them. We have to choose which vectors we pack considering that each vector $v_j \in \mathcal{V}$ has an associated profit $p(v_j) \in \mathbb{N}_{\geq 0}$. A packing of the vectors is a mapping $\sigma\colon \mathcal{V} \to \{1, \dots, C\} \cup \{\bot\}$ such that $\sum_{v_j \in \sigma^{-1}(i)} v_j^\ell \leq T^\ell$ holds for all $i \in \{1, \dots, C\}$ and $\ell \in \{1, \dots, d\}$, which means no container is over-packed. The objective is to find a packing with a maximum total profit of the packed items, that is, we want to maximize $\sum_{i=1}^{C} \sum_{v \in \sigma^{-1}(i)} p(v)$.

Conjoining and Over-the-Rainbow Matchings. We introduce two useful problems to tackle the questions mentioned above, namely Perfect Over-the-Rainbow Matching and Conjoining Matching. A *matching* in a graph G describes a set of edges $M \subseteq E(G)$ without common nodes, that is, $e_1 \cap e_2 = \emptyset$ for all distinct $e_1, e_2 \in M$. A matching is *perfect* if it covers all nodes. In the Perfect Over-the-Rainbow Matching problem, we are given an graph G as well as a *color function* $\lambda\colon E(G) \to 2^{\mathcal{C}} \setminus \{\emptyset\}$ which assigns a non-empty set of colors to each edge, and an integer ℓ. For each edge e and each color $c \in \lambda(e)$, there is a non-negative weight $\gamma(e, c)$. The objective is to find a perfect matching M and a surjective function $\xi\colon M \to \mathcal{C}$ with $\xi(e) \in \lambda(e)$ for each $e \in M$ such that $\sum_{e \in M} \gamma(e, \xi(e)) \leq \ell$. The surjectivity guarantees that each color must appear at least once. We call such a pair (M, ξ) a perfect *over-the-rainbow matching* and the term $\sum_{e \in M} \gamma(e, \xi(e))$ denotes its *weight*. This name comes from the closely related *rainbow matching* problem, where each color appears exactly once [21, 25]. In contrast to our problem, a sought rainbow matching covers as many colors as possible, but not necessarily all, and the maximum size of a rainbow matching is bounded by the number of colors. In our variant we must cover all colors, and likely have to cover some colors more than once to get a perfect matching. Formally, the problem is defined as follows:

Perfect Over-the-Rainbow Matching	*Parameter:* The number of colors $\lvert\mathcal{C}\rvert$
Input:	A graph G, a set of colors $\mathcal{C} = \{1, \dots, \lvert\mathcal{C}\rvert\}$, a function $\lambda\colon E \to 2^{\mathcal{C}} \setminus \{\emptyset\}$, edge weights $\gamma : \{(e, c) \mid e \in E(G), c \in \lambda(e)\} \to \mathbb{Q}_{\geq 0}$, and a number ℓ
Task:	Find a perfect over-the-rainbow matching (M, ξ) in G of weight at most ℓ.

We sometimes omit the surjective function ξ, if it is clear from the context.

Related to this problem is Conjoining Matching: We have a partition $V_1 \uplus \dots \uplus V_t$ of the nodes of G and a pattern graph H with $V(H) = \{V_1, \dots, V_t\}$. Instead of covering all colors in a perfect matching, this problems asks to find a *conjoining* matching $M \subseteq E(G)$, which is a perfect matching such that for each $\{V_i, V_j\} \in E(H)$ there is an edge in M with one node in V_i and the other in V_j. Roughly speaking, each edge in H corresponds to some edges in G of which at least one has to be taken by M. Formally, the problem is given by:

[2] This is due to the fact that in the multi-dimensional setting, we can simply model multiple occurrences of the same vector by introducing an additional dimension encoding the index of the vector. This is not possible in the one-dimensional case.

Conjoining Matching *Parameter:* The number of edges of H

Input: A weighted graph $G = (V, E, \gamma)$ with $\gamma\colon E \to \mathbb{Q}_{\geq 0}$, a node partition $V_1 \uplus \cdots \uplus V_t$, a number ℓ, and a graph H with $V(H) = \{V_1, \ldots, V_t\}$

Task: Find a perfect matching M in G of weight at most ℓ such that for each edge $\{V_i, V_j\} \in E(H)$ there is an edge $\{u, v\} \in M$ with $u \in V_i$ and $v \in V_j$.

Gutin et al. [16, Theorem 7] and Marx and Pilipczuk [27] gave randomized fixed-parameter algorithms for Conjoining Matching on loop-free graphs H. We show how a simple reduction also solves the problem on graphs with loops.

▶ **Lemma 5.** *The* Conjoining Matching *problem can be solved by a randomized algorithm (with bounded false negative rate in $n + \ell$) in time $2^{|E(H)|} \cdot n^{O(1)} \cdot \ell^{O(1)}$, even if H contains self-loops.*

Sketch of Proof. If H does not contain self-loops the claim is proven by Gutin et al. [16]. The case that H does contain self-loops can be reduced to the loop-free version by a simple layering argument: First direct the edges of H arbitrarily (for instance by using the lexicographical order of the nodes) and then define G' and H' as

$$\begin{aligned}
V(H') &= \{\, h', h'' \mid h \in V(H) \,\} \cup \{\, h^* \,\},\\
E(H') &= \{\, \{h'_i, h''_j\} \mid \{h_i, h_j\} \in E(H) \,\},\\
V(G') &= \{\, v', v'', v^* \mid v \in V(G) \,\},\\
E(G') &= \{\, \{v', v^*\}, \{v'', v^*\} \mid v \in V(G) \,\} \cup \{\, \{v', w''\} \mid \{v, w\} \in E(G) \,\}.
\end{aligned}$$

Observe that H' is loop-free, and $|E(H)| = |E(H')|$. Further note that, in any perfect matching in G', for each $v \in V(G)$ either v' or v'' must be matched with v^*; the other node together with its matching partner corresponds to an edge in a corresponding perfect matching in G as it is only connected to v^* or $\{w', w'' \mid \{v, w\} \in E(G)\}$. Finally, to preserve weights, set $\gamma'(\{v', v^*\}) = \gamma'(\{v'', v^*\}) = 0$ and $\gamma'(\{v', w''\} = \gamma(\{v, w\})$ for all $v, w \in V(G)$. ◀

3 Reducing Packing and Covering Problems to Finding Perfect Over-the-Rainbow Matchings

The first phase to solve these packing and covering problems is to interpret them as Perfect Over-the-Rainbow Matching problems. Each problem admits a similar procedure: Guess the packing of the small vectors; guess the number of large vectors for each container; use these guesses to pack the large vectors by formulating the problem as a matching problem in a graph. The idea is that the nodes of this graph represent the large vectors. An edge represents that both endpoints can be placed into the same container to satisfy the condition of the problem, i. e., either to fit into the container or to cover it. Introducing a color and a weight function for the edges, we manage to handle the containers already filled with some small vectors and the overall profits of the packing. Note that the guessing also serves as a transformation from the minimization and maximization problems to decision problems as each guess also corresponds to some fixed number of containers and if applicable to the profit. So we ask if there is a solution with these numbers and thus we can solve this question via a reduction.

Identifying the Set of Small Vectors. Before we can proceed as mentioned above, we first need to identify the sets $\mathcal{V}_L$ and $\mathcal{V}_S$ of large and small vectors explicitly. This can be done via a reduction to the 3-Hitting Set problem as follows: The set of elements is given by the set of vectors $\mathcal{V}$ and we compute all sets $S \subseteq \mathcal{V}$ of triplets that fit together in a single container, i. e., $|S| = 3$ and $\sum_{v \in S} v \leq T$. Consider a hitting set H for this instance. Then the set $\mathcal{V} \setminus H$ is large. To see this, consider any three distinct vectors $u, v, w \in \mathcal{V} \setminus H$. If we had $u + v + w \leq T$, then the set $\{u, v, w\}$ would be part of the computed selection of subsets. Yet, $\{u, v, w\} \cap H = \emptyset$ – a contradiction. We can pick the given number of small vectors k and use Fact 1 to obtain a hitting set $H \subseteq \mathcal{V}$ of size at most k. We set $\mathcal{V}_L = \mathcal{V} \setminus H$ and $\mathcal{V}_S = H$. As there are $O(n^3)$ sets of triplets this yields a run time of $2.27^k \cdot n^{O(1)}$ (see Fact 1).

The Case of Packing Vectors. Recall that in the Vector Packing problem we are given n vectors of dimension d and a set of containers, each with the same size limitation $T \in \mathbb{Q}^d$. Furthermore, we assume that the sets $\mathcal{V}_S$ and $\mathcal{V}_L$ are given explicitly by using the computation explained above. Any solution needs at most $|\mathcal{V}|$ and at least $\lceil |\mathcal{V}_L|/2 \rceil$ containers. Furthermore, if there is a solution with $m \leq |\mathcal{V}|$ there is also a solution with m' containers for any $m' \in \{m+1, \dots, |\mathcal{V}|\}$. Thus a binary search for the optimal number of containers between the given bounds is possible. Let C be the current guess of the number of containers. Now we have to decide whether there exists a solution using exactly C containers.

We guess the packing of the small vectors, that is, we try all possible partitions into at most $\min\{C, k\}$ subsets. It is not hard to see that the number of such partitions is upper bounded by the k^{th} Bell number: The first vector is packed by itself, the second can either be packed with the first one or also by itself, and so on. If any of the corresponding containers is already over-packed, we discard the guess. In the following, we call the used containers *partially filled* as some area is already occupied by small vectors. For these partially filled containers, we guess which of them are finalized, i. e., which of them do not contain an additional large vector in the optimal solution, and discard them for the following steps. There are at most 2^k such guesses. We denote the number of discarded containers as C_0. For each of the remaining partially filled containers, we introduce a new color. Furthermore, we introduce a color $\top$ representing the empty containers if existent. Hence, the resulting set of colors $\mathcal{C}$ has a cardinality of at most $k + 1$. For each $c \in \mathcal{C}$, we denote by $s(c) \in \mathbb{Q}^d$ the residual size in the corresponding container.

We place the large vectors $\mathcal{V}_L$ inside the $C - C_0$ residual containers by reducing it to a Perfect Over-the-Rainbow Matching problem. Note that if the current guesses are correct, each of the $C - C_0$ containers receives at least one and at most two large vectors. Hence, we may assume $|\mathcal{V}_L|/2 \leq (C - C_0) \leq |\mathcal{V}_L|$ (and reject the current guess otherwise). Furthermore, the number of containers receiving one or two large items, respectively, is already determined by C and C_0. We denote these numbers by C_1 and C_2 and remark that $C_2 = |\mathcal{V}_L| - (C - C_0) \geq 0$ and $C_1 := (C - C_0) - C_2 = 2(C - C_0) - |\mathcal{V}_L| \geq 0$.

We now construct a graph $G = (V, E)$ to find a feasible packing. Every large vector $v \in \mathcal{V}_L$ is represented by two nodes v and v' in V. Let $\mathcal{V}'_L = \{v' \mid v \in \mathcal{V}_L\}$. Next, we define a set $\mathcal{B}$ of $2 \cdot C_2$ new nodes called *blocker nodes*, which ensures that all vectors are placed inside exactly $(C - C_0)$ containers. We define $V := \mathcal{V}_L \cup \mathcal{V}'_L \cup \mathcal{B}$. In this graph, an edge between the nodes in $\mathcal{V}_L \cup \mathcal{V}'_L$ represents a possible packing of the large vectors inside one container. Hence, we add an edge $e = \{v, w\}$ between two original vectors $v, w \in \mathcal{V}_L$ and assign this edge some color $c \in \mathcal{C}$ if these vectors fit together inside the corresponding container. Furthermore, we add an edge between a vector $v \in \mathcal{V}_L$ and its copy $v' \in \mathcal{V}'_L$ and assign it the color $c \in \mathcal{C}$ if the vector alone fits inside the corresponding container. More formally, we introduce the set of edges $E_c := \{\{u, v\} \mid u, v \in \mathcal{V}_L, u + v \leq s(c)\} \cup \{\{v, v'\} \mid v \in \mathcal{V}_L, v \leq s(c)\}$ for each color $c \in \mathcal{C}$. Additionally, we introduce the edges of a complete bipartite graph between the copied

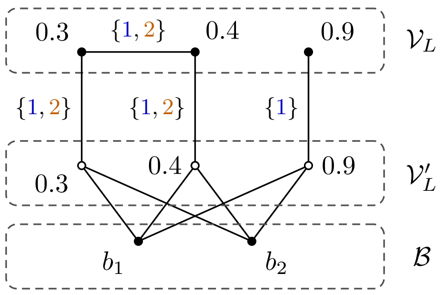

Figure 1 Construction of the graph G for a Bin Packing instance with sets $\mathcal{V}_S = \{0.1, 0.15, 0.2\}$ and $\mathcal{V}_L = \{0.3, 0.4, 0.9\}$. The guessed number of bins is $C = 3$. All small items are packed separately and the bin containing 0.15 is finalized ($C_0 = 1$). There is thus a bin containing 0.1 associated with color 1 (the first value in the braces) and a bin containing 0.2 associated with color 2 (the second value in the braces). The color $\perp$ used between all nodes of $\mathcal{V}_L'$ and all nodes of $\mathcal{B}$ are omitted.

nodes $\mathcal{V}_L'$ on the one hand and the blocker nodes $\mathcal{B}$ on the other hand. More formally, we define $E_\perp := \{\{v', b\} \mid v' \in \mathcal{V}_L', b \in \mathcal{B}\}$. Together, we get $E := E_\perp \cup \bigcup_{c \in \mathcal{C}} E_c$. Finally, we define the color function λ with $\lambda \colon E \to 2^{\mathcal{C} \cup \{\perp\}}$, such that each edge in E_c gets color c for each $c \in \mathcal{C}' := \mathcal{C} \cup \{\perp\}$. More formally, we define $\lambda(e) := \{c \in \mathcal{C} \cup \{\perp\} \mid e \in E_c\}$. See Figure 1 for an example of the construction. Note that the weights on the edges are irrelevant in this case and can be set to one, i. e. $\gamma(e, c) = 1$. To finalize the reduction, we have to define the size ℓ of the matching we are looking for. We aim to find a perfect matching and hence are searching for a matching of size $\ell := |\mathcal{V}_L| + C_2$. Note that if $C_2 = 0$ and therefore no blocker nodes are introduced, we also remove the color $\perp$ from the set of colors.

▶ **Lemma 6.** *There is a packing of the large vectors $\mathcal{V}_L$ inside $(C - C_0)$ containers such that each container holds at least one large vector if and only if the above described instance for* Perfect Over-The-Rainbow-Matching *is a "yes"-instance.*

Proof. Assume there is a packing of the vectors $\mathcal{V}_L$ inside $(C - C_0)$ containers such that each container holds at least one large vector. In this case, we can construct a perfect over-the-rainbow matching M as follows. For each pair of vectors $v, w \in \mathcal{V}_L$ that is assigned to the same container, we choose the corresponding edge $\{v, w\}$ for the matching and assign it the corresponding color $c \in \mathcal{C}$. For each vector $v \in \mathcal{V}_L$ that is the only large vector in its container, we choose the edge $\{v, v'\}$ for the matching and assign it the corresponding color $c \in \mathcal{C}$. To this point all the vectors in $\mathcal{V}_L$ are covered by exactly one matching edge since each of them is contained in exactly one container.

Note that in the given packing there have to be exactly $C_1 = 2(C - C_0) - |\mathcal{V}_L|$ containers with exactly one large vector and $C_2 = |\mathcal{V}_L| - (C - C_0)$ containers with exactly two large vectors. As a consequence, there are exactly $2 \cdot C_2$ nodes in $\mathcal{V}_L'$ that are not yet covered by a matching edge since their originals are covered by edges between each other. For each of these nodes, we choose an individual node from the set $\mathcal{B}$ and define the edge between these nodes as a matching edge and assign it the color $\perp$. Since there are exactly $2 \cdot C_2$ blocker nodes, we cover all nodes in V with matching edges and hence we have constructed a perfect matching. Each color $c \in \mathcal{C} \setminus \{\top\}$ is represented by one partially filled container and hence each has to appear in the matching. Moreover, if the color $\top$ was introduced, that is, there were less than $C - C_0$ containers partially covered by small vectors, then there was a container exclusively containing large vectors and hence $\top$ was used in the matching as well. Therefore, we indeed constructed a perfect over-the-rainbow matching.

Conversely, assume that we are given a perfect over-the-rainbow matching M. Consequently, each vector in $\mathcal{V}_L$ is covered by exactly one matching edge. As M contains at most $|\mathcal{V}_L| + C_2$ edges, and $2 \cdot C_2$ edges are needed to cover the nodes in $\mathcal{B}$, there are exactly $|\mathcal{V}_L| - C_2 = (C - C_0)$ matching edges containing the nodes from $\mathcal{V}_L$. As in M each color is present, we can represent each container by such a matching edge and place the corresponding vector or vectors inside corresponding containers. If a color $c \in \mathcal{C} \setminus \{\top\}$ appears more than once, we use an empty container for the corresponding large vectors. ◀

To decide if there is a packing into at most C containers, we find a partition of the k small vectors with $O(k!)$ guesses, and the to-be-discarded containers with $O(2^k)$ guesses. Constructing the graph G needs $O(n^2 k)$ operations. By Theorem 3, a perfect over-the-rainbow matching over $k + O(1)$ colors with weight $\ell \in O(n)$ can be computed in time $2^k \cdot n^{O(1)}$. To find the correct C we call the above algorithm in binary search fashion $O(\log(n))$ times, as we need at most n containers. This results in a run time of $2^{2k} \cdot k! \cdot n^{O(1)} = 4^k \cdot k! \cdot n^{O(1)}$.

4 Find Over-the-Rainbow Matchings with Conjoining Matchings

In the previous section, we reduced Vector Packing to the Perfect Over-the-Rainbow Matching problem. Of course, all this effort would be in vain without the means to find such matchings. This section presents a reduction to the task of finding a conjoining matching, which results in a parameterized algorithm for finding perfect over-the-rainbow matchings by applying Lemma 5. Overall, this proves Theorem 3, which is repeated below for convenience:

▷ Claim (Claim of Theorem 3). There is a randomized algorithm (with bounded false negative rate in $n + \ell$) that solves Perfect Over-The-Rainbow Matching in time $2^{|C|} \cdot n^{O(1)} \cdot \ell^{O(1)}$.

We aim to construct graphs H' and G' such that G has a perfect over-the-rainbow matching if, and only if, G' has a perfect conjoining matching with respect to H' of the same weight. Recall that in an over-the-rainbow matching, we request an edge of every color to be part of the matching; while in a conjoining matching, we request edges between certain sets of nodes to be part of the matching. For the reduction, we transform G into $G_1, \dots, G_{|\mathcal{C}|}$ where each G_c is a copy of G containing only edges of color c. Hence, $V(G_c) = \{v_c \mid v \in V(G)\}$, i.e., v_c is the copy of $v \in V(G)$ in $V(G_c)$, and G_c contains only edges $e \in E(G)$ with $c \in \lambda(e)$. We set G' to be the disjoint union of the G_c while setting $V(H') = \{\, V(G_c) \mid c \in \mathcal{C} \,\}$ and $E(H') = \{\, \{h, h\} \mid h \in V(H') \,\}$. Now a conjoined matching contains an edge of every color – however, the same edge of G could be used in multiple ways in the different copies G_c.

To address this issue, we introduce a gadget that will enforce any perfect matching in G' to use at most one copy of every edge of G. In detail, for every node $v \in V(G)$ we will add an independent set $J(v)$ of size $|\mathcal{C}| - 1$ to G'. Furthermore, we will fully connect $J(v)$ to all copies of v in G', that is, we add the edges $\{v_c, x\}$ for all $c \in \mathcal{C}$ and $x \in J(v)$ to G'. This construction is illustrated in Figure 2. Observe that in any perfect matching of G' all elements of $J(v)$ must be matched and, thus, we "knock-out" $|J(v)| = |\mathcal{C}| - 1$ copies of v in G' – leaving exactly one copy to be matched in one G_c. We add one more node to H' that represents the union of all the sets $J(v)$ and has no connecting edge. To complete the description of the reduction, let us describe the weight function of G': For each $e \in E(G')$, we define $\gamma'(e) = \gamma(e, c)$ if there exists a $c \in \mathcal{C}$ such that $e \in E(G_c)$, and $\gamma'(e) = 0$ otherwise. Note that this definition implies that $\gamma'(e) = 0$ for each e with $e \cap J(v) \neq \emptyset$ for some $v \in V(G)$.

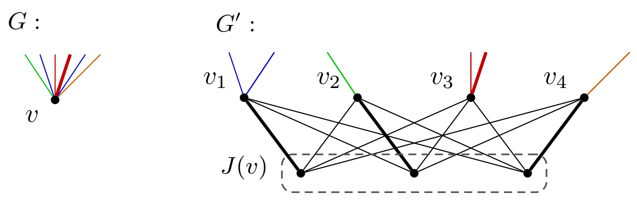

Figure 2 Reducing the problem of finding a perfect over-the-rainbow matching to the problem of finding a perfect conjoining matching. Left: single node of the colored input graph with a thick edge from a perfect matching. Right: $|\mathcal{C}| = 4$ copies of v in G'; the corresponding subgraphs G_c only contain edges of a single color. At the bottom, the added set $J(v)$ which is fully connected to all copies of v. The thick edges indicate how these nodes are paired in a perfect matching.

▶ **Lemma 7** ($\star$). *Let (G, λ, γ) be a colored and edge-weighted graph, and let G' and H' be defined as above. There is a perfect over-the-rainbow-matching M of weight ℓ in G if, and only if, there is a perfect conjoining matching M' of weight ℓ in G'.*

Proof of Theorem 3. Let $(G, \lambda, \gamma, \ell)$ be an instance of PERFECT OVER-THE-RAINBOW MATCHING. We construct in polynomial time an instance (G', H', γ', ℓ) of CONJOINING MATCHING, where the partition of $V(G')$ is defined as $V(G_1) \dot\cup V(G_2) \dot\cup \ldots \dot\cup V(G_{|\mathcal{C}|}) \dot\cup J$ with $J = \bigcup_{v \in V(G)} J(v)$ and $E(H')$ contains one self loop for each $V(G_c)$, $c \in \mathcal{C}$. By Lemma 7, (G', H', γ', ℓ) has a perfect conjoining matching of weight ℓ if, and only if $(G, \lambda, \gamma, \ell)$ has a perfect over-the-rainbow matching of weight ℓ. We apply Lemma 5 to find such a conjoining matching in time $2^{|E(H')|} \cdot n^{O(1)} \cdot \ell^{O(1)}$. Observe that $|E(H')| = |\mathcal{C}|$ and, thus, we can find the sought perfect over-the-rainbow matching in time $2^{|\mathcal{C}|} \cdot n^{O(1)} \cdot \ell^{O(1)}$. ◀

5 A Deterministic Algorithm for Bin Packing with Few Small Items

We now present a *fully-deterministic algorithm* for BIN PACKING. The price we have to pay for circumventing the randomness is an increased run time as we avoid the polynomial identity testing subroutine. On the bright side, this makes the algorithm straightforward and a lot simpler. We anticipate that extending this algorithm for VECTOR PACKING seems quite challenging. The main obstacle here is to identify the maximum item size in some sets, a task for which there does not seem to be a sensible equivalent notion for vectors.

About the Structure of Optimal Solutions. In the following, we prove the existence of an optimal solution that admits some useful properties regarding the placement of large items relating to small ones. These properties are utilized in the algorithm later on.

▷ Claim 8. There exists an optimal solution where the total size of small items on each bin containing only small items is larger than the total size of small items on each bin containing additionally large items.

▷ Claim 9. Given an optimal solution and an arbitrary order of the bins containing small items and exactly one large item. We can repack these large items correctly using a largest fitting approach with respect to the order of the bins. In detail, we place greedily the largest fitting item into the current bin.

▷ Claim 10. Consider an optimal solution where each partially filled bin contains exactly two items. Let i_s be the smallest large item and i_ℓ be the largest one and let them fit together inside a partially filled bin. Then there exists an optimal solution, where i_s is positioned inside a partially filled bin, together with the largest large item, that does fit additionally.

▷ Claim 11. Consider an instance I, where the largest large item i_ℓ does not fit together with the smallest large item i_s inside any partially filled bin and there is an optimal solution, where all partially filled bins contain exactly two large items. Then there is an optimal solution which places i_ℓ together with the largest fitting large item inside one bin, or i_ℓ is placed alone inside a bin, if there no large item fits together with i_ℓ inside one bin.

The Complete Algorithm. In the first step of the algorithm, we sort the items regarding their sizes in $O(n \log(n))$. Next, we guess the distribution of the small items. Since there are at most k small items, there are at most $O(k!)$ possible guesses. We call the bins containing small items partially filled bins. There are at most k of these bins.

Then, we guess a bin b_1 that does not contain any additional large item. All the partially filled bins, containing small items with a larger total size than b_1 do not contain any large item as well, see Claim 8. Thus we can discard them from the following considerations. There are at most k possibilities for the guess of b_1.

Now, we guess which of remaining partially filled bins only contain one large item. There are at most $O(2^k)$ possibilities. We consider all partially filled bins for which we guessed that they only contain one large item in any order and pair them with the largest fitting item. By Claim 8, we know that an optimal packing with this structure exists. Afterwards, we discard these bins from the following considerations.

It remains to pack the residual large items. Each residual, partially filled bin contains exactly two large items in the optimal solution, otherwise the guess was wrong. To place the correct large item, we proceed as follows: Iterate through the large items in non-ascending order regarding their sizes. Let i_ℓ be the currently considered item. Further, let i_s be the smallest large item from the set of large items that still need to be placed. Depending on the relation between i_ℓ and i_s, we place at least one of these two items inside a bin. For the first case, it holds that i_ℓ does not fit together with i_s inside a partially filled bin. Then, we place i_ℓ together with the largest fitting item i from the set of large items that are not already placed inside one empty bin or place it alone inside an empty bin if such an item does not exist. The item i can be found, or its non-existence be proved, in time $O(\log(n))$. For the second case, it holds that i_ℓ together with i_s does fit inside one partially filled bin. Then, we guess which partially filled bin contains i_s and place it inside this bin together with the largest unplaced item that fits inside this bin. The largest fitting item can be found in time $O(\log(n))$, and there are at most $O(k!)$ possible guesses total.

In the following, we argue that in both cases there exists an optimal solution where the items are placed exactly as the algorithm does assuming all the guesses are correct. When all the previous steps are correct, we can consider the residual set of items as a new instance, where there exists an optimal solution, where all partially filled bins contain exactly two large items (and we already know the correct distribution of small items). For this new instance we fill one bin correctly due to Claim 11 in Case 1. Since this bin is filled correctly with respect to an existing optimal solution, we again can consider the residual set of items as an independent instance that needs solving. On the other hand in Case 2, we know by Claim 10, that there exists an optimal solution for this reduced instance where i_s is placed together with the largest fitting large item inside one partially filled bin. If we guess this bin correctly, we have filled one bin correctly with regard to the considered instance. Hence when reducing the considered instance to the residual set of items (without this just filled bin) there exists an optimal solution for this instance with exactly one less bin.

After placing all the large items, we compare the obtained solution with the so far best solution, save it if it uses the smallest number of bins so far, and backtrack to the last decision. Since it iterates all possible guesses, this algorithm generates an optimal packing and its run time is bounded by $O((k!)^2 \cdot k \cdot 2^k \cdot n \log(n))$.

6 Conclusion and Further Work

We provided a randomized algorithm with one-sided error to identify perfect over-the-rainbow matchings. Via reductions to this problem, we obtained randomized $4^k \cdot k! \cdot n^{O(1)}$-time algorithms for the vector versions of Bin Packing, Multiple Knapsack, and Bin Covering parameterized by the number k of *small items.* We believe that studying this parameter is a natural step towards the investigation of the stronger parameterizations by the number of distinct item types. In that setting, the number of small items can then be large – however, there are only few small-item-types and, thus, we may hope to adapt some of the techniques developed in this article to this setting.

As a working horse we used a randomized algorithm to find conjoining matchings. As mentioned by Marx and Pilipczuk, it seems challenging to find such matchings by a deterministic fixed-parameter algorithm. Alternatively, we could search directly for deterministic algorithms for the problems as presented in this article. We present such an algorithm for Bin Packing; however, the techniques used in its design do not seem to generalize to the vector version.

References

1 N. Alon, Y. Azar, J. Csirik, L. Epstein, S. V. Sevastianov, A. P. A. Vestjens, and G. J. Woeginger. On-line and off-line approximation algorithms for vector covering problems. *Algorithmica*, 21(1):104–118, 1998.

2 L. Babel, B. Chen, H. Kellerer, and V. Kotov. Algorithms for on-line bin-packing problems with cardinality constraints. *Discrete Appl. Math.*, 143(1-3):238–251, 2004.

3 N. Bansal, M. Eliás, and A. Khan. Improved approximation for vector bin packing. In *Proc. SODA 2016*, pages 1561–1579, 2016.

4 H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Comput. Sci. Rev.*, 24:63–79, 2017.

5 J. Csirik, J. B. G. Frenk, M. Labbé, and S. Zhang. On the multidimensional vector bin packing. *Acta Cybern.*, 9(4):361–369, 1990.

6 M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms.* Springer, 2015.

7 L. Epstein, L. M. Favrholdt, and A. Levin. Online variable-sized bin packing with conflicts. *Discrete Opt.*, 8(2):333–343, 2011.

8 L. Epstein, A. Levin, and R. van Stee. Approximation schemes for packing splittable items with cardinality constraints. *Algorithmica*, 62(1-2):102–129, 2012.

9 S. Fafianie and S. Kratsch. A shortcut to (sun)flowers: Kernels in logarithmic space or linear time. In *Proc. MFCS 2015*, volume 9235 of *Lecture Notes Comput. Sci.*, pages 299–310, 2015.

10 M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao. Resource constrained scheduling as generalized bin packing. *J. Combinatorial Theory, Ser. A*, 21(3):257–298, 1976.

11 M. R. Garey and D. S. Johnson. Approximation algorithms for bin packing problems: A survey. In *Analysis and design of algorithms in combinatorial optimization.* Springer, 1981.

12 S. Geng and L. Zhang. The complexity of the 0/1 multi-knapsack problem. *J. Comput. Sci. Tech.*, 1(1):46–50, 1986.

13 M. X. Goemans and T. Rothvoß. Polynomiality for bin packing with a constant number of item types. In *Proc. SODA 2014*, pages 830–839, 2014.

14 T. F. Gonzalez. *Handbook of approximation algorithms and metaheuristics.* Chapman and Hall/CRC, 2007.

15 J. Guo, F. Hüffner, and R. Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proc. IWPEC 2004*, volume 3162 of *Lecture Notes Comput. Science*, pages 162–173, 2004.

16 G. Z. Gutin, M. Wahlström, and A. Yeo. Rural postman parameterized by the number of components of required edges. *J. Comput. Syst. Sci.*, 83(1):121–131, 2017.

17 R. Hoberg and T. Rothvoß. A logarithmic additive integrality gap for bin packing. In *Proc. SODA 2017*, pages 2616–2625, 2017.

18 K. Jansen and K. Klein. About the structure of the integer cone and its application to bin packing. In *Proc. SODA 2017*, pages 1571–1581, 2017.

19 K. Jansen, S. Kratsch, D. Marx, and I. Schlotter. Bin packing with fixed number of bins revisited. *J. Comput. Syst. Sci.*, 79(1):39–49, 2013.

20 K. Jansen and R. Solis-Oba. A polynomial time $OPT + 1$ algorithm for the cutting stock problem with a constant number of object lengths. *Math. Oper. Res.*, 36(4):743–753, 2011.

21 M. Kano and X. Li. Monochromatic and heterochromatic subgraphs in edge-colored graphs–A survey. *Graphs and Combinatorics*, 24(4):237–263, 2008.

22 H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

23 C. Kenyon. Best-fit bin-packing with random order. In *Proc. SODA 1996*, 1996.

24 J. Kuipers. Bin packing games. *Math. Meth. Oper. Res.*, 47(3):499–510, 1998.

25 V. B. Le and F. Pfender. Complexity results for rainbow matchings. *Theor. Comput. Sci.*, 524:27–33, 2014.

26 S. Martello and P. Toth. *Knapsack problems*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Ltd., Chichester, 1990. Algorithms and computer implementations.

27 D. Marx and M. Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In *Proc. STACS 2014*, volume 25 of *Leibniz Int. Proc. Informatics*, pages 542–553, 2014.

28 S. Th. McCormick, S. R. Smallwood, and F. C. R. Spieksma. A polynomial algorithm for multiprocessor scheduling with two job lengths. In *Proc. SODA 1997*, pages 509—-517, 1997.

29 S. Th. McCormick, S. R. Smallwood, and F. C. R. Spieksma. A polynomial algorithm for multiprocessor scheduling with two job lengths. *Math. Oper. Res.*, 26(1):31–49, 2001.

30 M. Mnich and R. van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Comput. & OR*, 100:254–261, 2018.

31 K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

32 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

33 R. Niedermeier and P. Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *J. Discrete Algorithms*, 1(1):89–102, 2003.

34 P. W. Shor. *Random planar matching and bin packing*. PhD thesis, Massachusetts Institute of Technology, Department of Mathematics, 1985.

35 P. W. Shor. The average-case analysis of some on-line algorithms for bin packing. *Combinatorica*, 6(2):179–200, 1986.

36 M. Sorge, R. van Bevern, R. Niedermeier, and M. Weller. A new view on rural postman based on Eulerian extension and matching. *J. Discrete Algorithms*, 16:12–33, 2012.

37 R. van Bevern. Towards optimal and expressive kernelization for d-hitting set. *Algorithmica*, 70(1):129–147, 2014.

38 G. J. Woeginger. There is no asymptotic PTAS for two-dimensional vector packing. *Inf. Process. Lett.*, 64(6):293–297, 1997.