Decomposing a graph into subgraphs with small components

Rain Jiang ☉ Kai Jiang ☉ Minghui Jiang ☉*
Home School, USA

Abstract

The component size of a graph is the maximum number of edges in any connected component of the graph. Given a graph $G$ and two integers $k$ and $c$, $(k, c)$-Decomposition is the problem of deciding whether $G$ admits an edge partition into $k$ subgraphs with component size at most $c$. We prove that for any fixed $k \geq 2$ and $c \geq 2$, $(k, c)$-Decomposition is NP-complete in bipartite graphs. Also, when both $k$ and $c$ are part of the input, $(k, c)$-Decomposition is NP-complete even in trees. Moreover, $(k, c)$-Decomposition in trees is W[1]-hard with parameter $k$, and is FPT with parameter $c$. In addition, we present approximation algorithms for decomposing a tree either into the minimum number of subgraphs with component size at most $c$, or into $k$ subgraphs minimizing the maximum component size. En route to these results, we also obtain a fixed-parameter algorithm for Bin Packing with the bin capacity as parameter.

1 Introduction

Edge coloring is the problem of assigning a color to each edge of a graph such that no two adjacent edges have the same color. Any edge coloring of a graph with maximum degree $\Delta$ requires at least $\Delta$ colors. Vizing’s theorem states that $\Delta + 1$ colors suffice. The minimum number of colors in any edge coloring of a graph is its chromatic index. Deciding whether the chromatic index of a graph with maximum degree $\Delta$ is $\Delta$ or $\Delta + 1$ is generally a hard computational problem, although a constructive proof of Vizing’s theorem [25] implies a polynomial-time approximation algorithm with additive error at most 1. Holyer [15] proved that edge coloring cubic graphs with three colors, where $\Delta = 3$, is already NP-complete. Extending this result, Leven and Galil [21] proved that edge coloring $\Delta$-regular graphs with $\Delta$ colors is NP-complete for any $\Delta \geq 3$. Cai and Ellis [3] further proved that edge coloring $\Delta$-regular line graphs of bipartite graphs with $\Delta$ colors is NP-complete for any odd $\Delta \geq 3$. On the other hand, it is known that $\Delta$ colors always suffice for edge coloring bipartite graphs; see [4, Proposition 18.1.3] for a simple proof. Moreover, edge coloring bipartite graphs (or bipartite multigraphs) admits an efficient polynomial-time exact algorithm, even when $\Delta$ is part of the input [6].

Define the component size of a graph as the maximum number of edges in any connected component of the graph. In this paper, we study the following problem that generalizes edge coloring:

Definition 1. Given a graph $G$ and two integers $k \geq 2$ and $c \geq 1$, $(k, c)$-DECOMPOSITION is the problem of deciding whether $G$ admits an edge partition into $k$ subgraphs with component size at most $c$.

We show that graph decomposition becomes hard, even in bipartite graphs, when the upper bound on the component size of the resulting subgraphs is relaxed from 1 (as in edge coloring) to any fixed $c \geq 2$:

Theorem 1. For any fixed $k \geq 2$ and $c \geq 2$, $(k, c)$-DECOMPOSITION is NP-complete in bipartite graphs.

* dr.minghui.jiang at gmail.com
The arboricity of a graph is the minimum number of parts in an edge partition of the graph such that each part induces a forest [26]. The trees in the forests may be restricted to various subclasses of trees, leading to several related concepts of arboricity. For example, the linear arboricity (respectively, star arboricity) of a graph is the minimum number of parts in an edge partition of the graph such that each part induces a disjoint union of paths (respectively, stars). While the arboricity of any graph can be computed in polynomial time [11], the arboricities with various restrictions on the trees often turn out to be hard to compute [17]. Our proof of Theorem 1 shows that arboricity and star arboricity with bounded component size are also hard to compute:

**Corollary 1.** For any fixed \( k \geq 2 \) and \( c \geq 2 \), deciding whether a bipartite graph admits an edge partition into \( k \) forests with component size at most \( c \) is NP-complete.

**Corollary 2.** For any fixed \( k \geq 2 \) and \( c \geq 2 \), deciding whether a bipartite graph admits an edge partition into \( k \) forests of stars with component size at most \( c \) is NP-complete.

The linear \( c \)-arboricity of a graph is the minimum number of parts in an edge partition of the graph into linear \( c \)-forests, i.e., disjoint unions of paths of length at most \( c \) [1]. Note that linear 1-arboricity is simply chromatic index. Thus the aforementioned results on edge coloring [15, 21, 3] imply that determining the linear 1-arboricity of a graph is hard. Bermond, Fouquet, Habib, and Peroche [1] proved that deciding whether a cubic graph has linear 3-arboricity at most 2 is NP-complete, and conjectured that determining the linear \( c \)-arboricity of a graph is hard for all \( c \). Since linear 2-arboricity is equivalent to arboricity and star arboricity with component size at most 2, the \( c = 2 \) case of this conjecture is confirmed by our Corollary 1 and Corollary 2:

**Corollary 3.** For any fixed \( k \geq 2 \), deciding whether a bipartite graph has linear 2-arboricity \( k \) is NP-complete.

For two graphs \( G \) and \( H \), an \( H \)-decomposition of \( G \) is an edge partition of \( G \) into subgraphs isomorphic to \( H \). For any fixed graph \( H \), the problem of deciding whether an input graph \( G \) admits an \( H \)-decomposition is NP-complete when \( H \) has a component of at least three edges [7], and is polynomially solvable when every component of \( H \) has at most two edges [2], in contrast to Corollary 3.

We established in Theorem 1 that \((k, c)\)-DECOMPOSITION is NP-complete in bipartite graphs. It is natural to ask whether the problem remains hard in simpler graphs. Our next theorem characterizes the complexity of \((k, c)\)-DECOMPOSITION in trees:

**Theorem 2.** When both \( k \) and \( c \) are part of the input, \((k, c)\)-DECOMPOSITION in trees is NP-complete, is W[1]-hard with parameter \( k \), and is FPT with parameter \( c \).

Theorem 2 implies that for any fixed \( c \), \((k, c)\)-DECOMPOSITION in trees can be solved in polynomial time. For the related arboricity problem of deciding whether a tree admits an edge partition into \( k \) linear \( c \)-forests, Chang, Chen, Fu, and Huang [5] presented an algorithm that runs in polynomial time when \( c \) is fixed.

The decision problem \((k, c)\)-DECOMPOSITION may be extended to two different optimization problems:

**Definition 2.** Given a graph \( G \) and an integer \( c \geq 1 \), MINIMUM-COLOR DECOMPOSITION is the problem of decomposing \( G \) into the minimum number of subgraphs with component size at most \( c \).

**Definition 3.** Given a graph \( G \) and an integer \( k \geq 2 \), MINIMUM-SIZE DECOMPOSITION is the problem of decomposing \( G \) into \( k \) subgraphs minimizing the maximum component size.
Theorem 1 implies that \textsc{Minimum-Color Decomposition} for any fixed \( c \geq 2 \) and \textsc{Minimum-Size Decomposition} for any fixed \( k \geq 2 \) are APX-hard and inapproximable within a factor of \( 3/2 \) in bipartite graphs. The previous results on edge coloring \([15, 21, 3]\) imply that \textsc{Minimum-Color Decomposition} for \( c = 1 \) is APX-hard and inapproximable within a factor of \( 4/3 \) in general graphs. In the next two theorems, we show that both minimization problems can be approximated very well in trees:

**Theorem 3.** There is a linear-time approximation algorithm for \textsc{Minimum-Color Decomposition} in trees that finds an edge partition of any tree \( T \) into at most \( k^* + 1 \) subgraphs with component size at most \( c \), where \( k^* \) is the smallest number such that \( T \) can be decomposed into \( k^* \) subgraphs with component size at most \( c \).

**Theorem 4.** There is a polynomial-time approximation scheme for \textsc{Minimum-Size Decomposition} in trees that finds an edge partition of any tree \( T \) into \( k \) subgraphs with component size at most \( (1 + \epsilon) c^* \), where \( c^* \) is the smallest number such that \( T \) can be decomposed into \( k \) subgraphs with component size at most \( c^* \).

Kleinberg, Motwani, Raghavan, and Venkatasubramanian [19] introduced \((k, c)\)-\textsc{Fragmented Coloring}, the problem of vertex partitioning a graph into \( k \) induced subgraphs such that the number of vertices in each component of each subgraph is at most \( c \). Our problem \((k, c)\)-\textsc{Decomposition} is equivalent to \((k, c)\)-\textsc{Fragmented Coloring} in line graphs, since edge partitioning a graph \( G \) is the same as vertex partitioning the line graph \( L(G) \).

The problem \((k, c)\)-\textsc{Fragmented Coloring} in general graphs is easily shown to be NP-hard for any \( k \geq 2 \) and \( c \geq 2 \), because the graph property of at most \( c \) vertices in each component is additive and induced-hereditary, while vertex partitioning into fixed additive induced-hereditary properties is NP-hard in general \([10]\), except the special case of bipartite testing, i.e., vertex partitioning into two independent sets. Also, the problem is trivial in bipartite graphs (and hence in trees), since every bipartite graph has a vertex partition into two induced subgraphs in which every component has only one vertex.

Our proofs of Theorem 2, Theorem 3, and Theorem 4 are based on a close relationship between \((k, c)\)-\textsc{Decomposition} in trees and the classical problem of \textsc{Bin Packing}:

**Definition 4.** Given \( n \) items of integer weights \( w_i, 1 \leq i \leq n \), and \( k \) bins of integer capacity \( c \), \textsc{Bin Packing} is the problem of deciding whether the \( n \) items can be packed into the \( k \) bins, that is, whether the set of \( n \) items can be partitioned into \( k \) subsets, such that the total weight of the items in each subset is at most \( c \). \textsc{Unary Bin Packing} is the version of \textsc{Bin Packing} where all integers in the problem instance are encoded in unary.

It is well-known that \textsc{Bin Packing} is strongly NP-hard \([12]\), and hence \textsc{Unary Bin Packing} is NP-hard, when both \( k \) and \( c \) are part of the input. In terms of parameterized complexity, Jansen et al. [16] proved that \textsc{Unary Bin Packing}, even with the condition \( \sum_{i=1}^{n} w_i = kc \), is already \( W[1] \)-hard with parameter \( k \). In the following theorem, we obtain a fixed-parameter algorithm for \textsc{Bin Packing} with parameter \( c \):

**Theorem 5.** \textsc{Bin Packing} admits an exact algorithm running in \( 2^{O(c^{3/2})} N^{O(1)} \) time, where \( N \) is the size of the problem instance encoded in binary.

Hochbaum and Shmoys [13] introduced the notion of \textit{dual approximation algorithms} that approximate the feasibility of a problem rather than its optimality. The existence of dual approximation algorithms for \textsc{Bin Packing}, similar to the one in the following theorem, are known \([14, \text{Sections 9.3.2.1 and 9.3.2.2}]\). We obtain a simple alternative via Theorem 5.
Theorem 6. Bin Packing admits a dual approximation algorithm that, given \( n \) items and \( k \) bins of capacity \( c \), and given \( 0 < \epsilon \leq 1 \), either decides that the \( n \) items can be packed into \( k \) enlarged bins of capacity \((1+\epsilon)c\) and returns yes, or decides that the \( n \) items cannot be packed into \( k \) bins of capacity \( c \) and returns no, in \( 2^{O(c^{-3})}N^{O(1)} \) time, where \( N \) is the size of the problem instance encoded in binary.

Throughout the paper, the size of a graph is the number of edges in it. We may refer to an edge partition of a graph into \( k \) subgraphs and the corresponding \( k \)-color assignment to the edges interchangeably.

2 Hardness of decomposing bipartite graphs

In this section we prove Theorem 1, Corollary 1, and Corollary 2. Fix any \( k \geq 2 \) and \( c \geq 2 \). The problem \((k, c)\)-DECOMPOSITION and the related arboricity problems are clearly in NP. We prove their NP-hardness in bipartite graphs in the following.

2.1 Building blocks \( H_i \)

The building blocks of our construction are a series of bipartite graphs \( H_i = H_i(k, c) \) for \( i \geq 0 \). \( H_0 \) is the star \( K_{1,kc} \) with one edge designated as an outlet. For \( i > 0 \), \( H_i \) has \( c^i \) outlets and is constructed recursively as follows. Take \( k-1 \) disjoint copies of \( H_{i-1} \) and \( c^{i-1} \) disjoint copies of \( K_{1,c} \). Attach a distinct outlet from each copy of \( H_{i-1} \) to the center of each copy of \( K_{1,c} \). Then designate the edges of all copies of \( K_{1,c} \) in the resulting graph \( H_i \) as its outlets. Refer to Figures 1 and 2 for some examples.

It is easy to verify that \( H_i \) is bipartite. Since it includes \((k-1)^i \) copies of \( K_{1,kc} \) and \( \sum_{j=1}^{i} c^{j-1}(k-1)^{i-j} \) copies of \( K_{1,c} \), its size is at most

\[
(k-1)^i kc + \sum_{j=1}^{i} c^j (k-1)^{i-j} \leq (k-1)^i kc + (c+k)^i \leq (c+k)^i(kc+1) \leq (c+k)^{i+2}.
\]  

(1)

Figure 1: \( H_0, H_1, \) and \( H_2 \) for \( k = 2 \) and \( c = 3 \).

Figure 2: \( H_2 \) for \( k = 3 \) and \( c = 2 \).
Lemma 1. \( H_i \) admits an edge partition into \( k \) subgraphs with component size at most \( c \). Moreover, in any edge partition of \( H_i \) into \( k \) subgraphs with component size at most \( c \), each component must be a star \( K_{1,c} \), and the \( c^i \) outlets must belong to the same subgraph.

Proof. We prove the lemma by induction on \( i \). For the base case when \( i = 0 \), the lemma clearly holds for \( H_0 = K_{1, kc} \). Now let \( i \geq 1 \) for the inductive step.

We first show that \( H_i \) admits an edge partition into \( k \) subgraphs with component size at most \( c \). By the induction hypothesis, \( H_{i-1} \) admits an edge partition into \( k \) subgraphs in which each component is a star \( K_{1,c} \), and the \( c^{i-1} \) outlets are in the same subgraph. Consider the \( k \)-color assignment to the edges of \( H_{i-1} \) corresponding to this edge partition. Then by rotation of colors, there is an edge partition of the union of the \( k-1 \) copies of \( H_{i-1} \) in \( H_i \), such that the outlets from different copies have different colors. Thus the outlets from the \( k-1 \) copies of \( H_{i-1} \) have \( k-1 \) different colors. Assign the remaining color to all edges of the \( c^{i-1} \) disjoint copies of \( K_{1,c} \) in \( H_i \). This color assignment corresponds to an edge partition of \( H_i \) into \( k \) subgraphs in which each component is a star \( K_{1,c} \), and the \( c^i \) outlets are all in the same subgraph.

We next show that in any edge partition of \( H_i \) into \( k \) subgraphs with component size at most \( c \), each component must be a star \( K_{1,c} \), and the \( c^i \) outlets must be in the same subgraph. Consider an arbitrary edge partition of \( H_i \) into \( k \) subgraphs with at most \( c \) edges in each component, and the corresponding \( k \)-color assignment to the edges of \( H_i \). Then in the derived edge partition of each copy of \( H_{i-1} \) in \( H_i \), each component is a star \( K_{1,c} \), and all \( c^{i-1} \) outlets are in the same subgraph, by the induction hypothesis. Moreover, since each component in each copy of \( H_{i-1} \) already has the maximum allowable size \( c \), the outlets from different copies of \( H_{i-1} \) must have different colors to avoid forming larger components. Then all edges of the \( c^{i-1} \) disjoint copies of \( K_{1,c} \) in \( H_i \) must have the only remaining color.

\[ \square \]

2.2 Reduction for \( k = 2 \)

Our proof for the \( k = 2 \) case of Theorem 1, Corollary 1, and Corollary 2 is based on a reduction from the NP-complete problem \( 2\)-COLORABILITY OF 3-UNIFORM HYPERGRAPHS [23].

A \( q \)-uniform hypergraph consists of a set \( V \) of vertices and a set \( E \) of hyperedges, where each hyperedge in \( E \) is a subset of exactly \( q \) vertices in \( V \). A hypergraph is \( k \)-colorable if its vertices can be colored with \( k \) colors such that in each hyperedge, not all \( q \) vertices have the same color.

Given a 3-uniform hypergraph \( G \), we will construct a bipartite graph \( G_2 \) with maximum degree \( 2c \) such that \( G \) is 2-colorable if and only if \( G_2 \) admits an edge partition into 2 subgraphs with component size at most \( c \).

Let \( \Delta \) be the maximum degree of \( G \), that is, the maximum number of hyperedges in which a vertex is contained. Let \( j \) be the smallest integer such that \( c^j \geq (c-1)\Delta \).

Include in \( G_2 \) a distinct copy of \( H_j \) for each vertex in \( G \), and a distinct copy of \( K_{1, c+1} \) for each hyperedge in \( G \). For each hyperedge in \( G \), which consists of three vertices, take \( c + 1 \) distinct outlets from the corresponding three copies of \( H_j \), with at least one outlet from each copy, then attach the \( c + 1 \) outlets to the \( c + 1 \) leaves of the copy of \( K_{1, c+1} \) corresponding to the hyperedge, with one outlet to each leaf. This completes the construction of \( G_2 \).

Recall that \( H_j \) has \( c^j \) outlets. Since \( c^j \geq (c-1)\Delta \), each copy of \( H_j \) can contribute at least \( c-1 \) distinct outlets to each adjacent copy of \( K_{1, c+1} \). On the other hand, of the three copies of \( H_j \) adjacent to each copy of \( K_{1, c+1} \), each must contribute at least 1 and hence at most \( c+1-2 = c-1 \) of the \( c+1 \) outlets. Thus there are enough outlets from each copy of \( H_j \) to the adjacent copies of \( K_{1, c+1} \).

By our choice of \( j \), we have \( c^{j-1} < (c-1)\Delta \) and hence \( c^j < c^2 \Delta \). Recall (1) that the size of \( H_j \) is at most \( (c + k)^{j+2} \). Since \( k = 2 \) and \( c \geq 2 \), we have

\[
(c + k)^{j+2} \leq (c^2)^{j+2} = c^{4j} < c^{4}(c^2 \Delta)^2 = c^{8} \Delta^2.
\]
The reduction is clearly polynomial. The following lemma completes the proof for the $k = 2$ case of Theorem 1, Corollary 1 and Corollary 2:

**Lemma 2.** $G$ is 2-colorable if and only if $G_2$ admits an edge partition into two subgraphs with component size at most $c$, if and only if $G_2$ admits an edge partition into two forests with component size at most $c$, if and only if $G_2$ admits an edge partition into two forests of stars with component size at most $c$.

**Proof.** It suffices to prove a cycle of four implications:

1. if $G$ is 2-colorable, then $G_2$ admits an edge partition into two forests of stars with component size at most $c$,
2. if $G_2$ admits an edge partition into two forests of stars with component size at most $c$, then $G_2$ admits an edge partition into two forests with component size at most $c$,
3. if $G_2$ admits an edge partition into two forests with component size at most $c$, then $G_2$ admits an edge partition into two subgraphs with component size at most $c$,
4. if $G_2$ admits an edge partition into two subgraphs with component size at most $c$, then $G$ is 2-colorable.

We first prove implication 1. Suppose that there is a 2-coloring of the vertices of $G$ such that in each hyperedge, not all three vertices have the same color. Color the edges in each copy of $H_j$ in $G_2$ with two colors to partition them into two subgraphs as in Lemma 1, such that all $c^j$ outlets in each copy of $H_j$ have the same color as the corresponding vertex in $G$. Next color the edges in each copy of $K_{1,c+1}$ in $G_2$ corresponding to a hyperedge in $G$, such that each edge of $K_{1,c+1}$ has a color different (note that there are only two colors available) from the adjacent outlet. Since in each hyperedge of $G$ not all three vertices have the same color, and since the $c + 1$ outlets adjacent to the $c + 1$ edges of $K_{1,c+1}$ include at least one outlet from each of the three copies of $H_j$, it follows that not all $c + 1$ edges have the same color. Thus we obtain an edge partition of $G_2$ into two subgraphs with component size at most $c$. Note that the two subgraphs are indeed two forests of stars.

Implications 2 and 3 are trivial.

We next prove implication 4. Suppose that $G_2$ admits an edge partition into two subgraphs with component size at most $c$. Color the edges of $G_2$ with two colors according to this partition. Then it follows by Lemma 1 that in each copy of $H_j$ in $G_2$, each component is a star with $c$ edges, and all outlets have the same color. Assign this color to the corresponding vertex in $G$. Since each outlet of $H_j$ is already in a component of the maximum allowable size $c$, each edge of $K_{1,c+1}$ must have a color different from its adjacent outlet. In each copy of $K_{1,c+1}$, not all $c + 1$ edges can have the same color under the constraint of component size at most $c$. Correspondingly, in each hyperedge of $G$, not all three vertices have the same color either. Thus $G$ is 2-colorable.

\[\square\]

### 2.3 Reduction for $k \geq 3$

Our proof for the $k \geq 3$ case of Theorem 1, Corollary 1, and Corollary 2 is based on a reduction from the NP-complete problem $k$-COLORABILITY, which asks whether a given graph $G$ admits a vertex coloring with $k$ colors. Maffray and Preissmann [24] proved that for any $k \geq 3$, $k$-COLORABILITY is NP-hard even in triangle-free graphs. Emden-Weinert, Hougardy, and Kreuter [9] proved that for any $k \geq 3$, $k$-COLORABILITY is NP-hard in graphs with maximum degree $k + \lceil \sqrt{k} \rceil - 1$. Note that $k$-COLORABILITY for $k = 2$ is just bipartite testing which is well-known to be solvable in polynomial time. Thus we had to prove the $k = 2$ case by reduction from a different problem.

Let $G$ be an input graph for $k$-COLORABILITY. Construct $G_k$ as follows. Let $j$ be the smallest integer such that $c^j$ is at least the maximum degree of $G$. For each vertex in $G$, include in $G_k$ a distinct copy of
$H_j$. For each edge in $G$, which consists of two vertices, take a distinct outlet from the copy of $H_j$ for each vertex, then join the two ends of the two outlets into one vertex.

It is easy to verify that $G_k$ is bipartite, and the reduction is polynomial. Moreover, by a similar argument as in the proof of Lemma 2, $G$ is $k$-colorable if and only if $G_k$ admits an edge partition into $k$ subgraphs with component size at most $c$, if and only if $G_k$ admits an edge partition into $k$ forests with component size at most $c$, if and only if $G_k$ admits an edge partition into $k$ forests of stars with component size at most $c$.

This completes the proof of Theorem 1, Corollary 1, and Corollary 2.

3 Fixed-parameter algorithm for bin packing

In this section we prove Theorem 5 by obtaining a fixed-parameter algorithm for BIN PACKING with parameter $c$.

Recall Definition 4 that the input to BIN PACKING consists of $n$ items of integer weights $w_i$, $1 \leq i \leq n$, and $k$ bins of integer capacity $c$. The problem has a solution only if $\max \{w_i \mid 1 \leq i \leq n\} \leq c$ and $\sum_{i=1}^{n} w_i \leq kc$, which we assume. Let $N$ be the size of the problem instance encoded in binary. Then $N = \Theta(n + \sum_{i=1}^{n} \log w_i + \log k + \log c)$. In particular, we have $\log kc = O(N)$.

For $1 \leq w \leq c$, let $a_w$ be the number of items with weight $w$. In $\mathcal{O}(n)$ time, we can compute $a_w$ for all $w$. Then $\sum_{w=1}^{c} w a_w = \sum_{i=1}^{n} w_i \leq kc$. With the multiplicities $a_w$ for $1 \leq w \leq c$, we no longer need $w_i$ for $1 \leq i \leq n$. Henceforth, we work on the $c$ integers $a_w$ only. Without loss of generality, we increase $a_1$ until $\sum_{w=1}^{c} w a_w = kc$. Note that each $a_w$ has magnitude at most $kc$, where $\log kc = O(N)$. It remains to decide whether the items with multiplicities $a_w$ can be partitioned into $k$ subsets, each of total weight exactly $c$.

Consider the two vectors $w = (1, 2, \ldots, c)$ and $a = (a_1, a_2, \ldots, a_c)$ in $\mathbb{Z}_{\geq 0}^c$. Then the condition $\sum_{w=1}^{c} w a_w = kc$ becomes $w \cdot a = kc$. For $b \geq 0$, let

$$\mathcal{P}_b = \{ x \in \mathbb{Z}_{\geq 0}^c \mid w \cdot x = b \}.$$ 

Then $\mathcal{P}_c$ is the set of vectors corresponding to all feasible patterns of item multiplicities for a single bin, and we can reformulate the problem as deciding whether there exists a multiset of vectors in $\mathcal{P}_c$ whose sum is $a$.

Recall that the partition number $p(n)$ for $n \geq 0$ is the number of multisets of positive integers summing up to $n$. In particular, $p(0) = 1$. Then, by definition, the size of $\mathcal{P}_c$ is exactly $p(c)$. Moreover, the size of $\mathcal{P}_b$ is at most $p(b)$ for all $b \geq 0$.

It is known that $p(n) = 2^{O(\sqrt{n})}$ [22, Theorem 15.7]. Let $d(c) \geq 0$ be the smallest integer such that $2^d > \sum_{b=0}^{d} p(hc)$ for all $d > d(c)$. Since $\sum_{b=0}^{d} p(hc) = 2^{O(\sqrt{d}c)}$, we have $d(c) = O(c)$.

Let $v_i$, $1 \leq i \leq p(c)$, be the vectors in $\mathcal{P}_c$. For any index set $I \subseteq \{1, \ldots, p(c)\}$, the vector $\sum_{i \in I} \lambda_i v_i$, where $\lambda_i \in \mathbb{Z}_{\geq 0}$ for all $i \in I$, is called an integer conical combination of vectors in $\mathcal{P}_c$. Consider the integer conical hull of $\mathcal{P}_c$ consisting of all integer conical combinations of vectors in $\mathcal{P}_c$:

$$\text{int.cone}(\mathcal{P}_c) = \left\{ \sum_{i=1}^{p(c)} \lambda_i v_i \mid \lambda_i \in \mathbb{Z}_{\geq 0} \text{ for } 1 \leq i \leq p(c) \right\}.$$ 

We prove the following lemma using a common technique for Carathéodory bounds; see for example [8, Lemma 3].

**Lemma 3.** Every vector in $\text{int.cone}(\mathcal{P}_c)$ can be represented as the integer conical combination of at most $d(c)$ distinct vectors in $\mathcal{P}_c$. 

7
Proof. Let $I$ be any set of $d$ distinct indices from $\{1, \ldots, p(c)\}$, where $0 \leq d \leq p(c)$. Then $I$ has $2^d$ distinct subsets. Denote by $|H|$ the size of a set $H$. For each subset $H \subseteq I$, where $0 \leq |H| \leq d$, the vector $x = \sum_{i \in H} v_i$ satisfies

$$w \cdot x = w \cdot \sum_{i \in H} v_i = \sum_{i \in H} w \cdot v_i = \sum_{i \in H} c = |H| c,$$

and hence $x \in \mathcal{P}_{hc}$ with $h = |H|$. Thus $x$ is one of the vectors in $\bigcup_{h=0}^{d} \mathcal{P}_{hc}$.

Consider any vector $u = \sum_{i \in I} \lambda_i v_i$, where $\lambda_i \in \mathbb{Z}_{\geq 0}$ for all $i \in I$. Suppose that $d > d(c)$. Then by our choice of $d(c)$, we have $2^d > \sum_{h=0}^{d} p(hc)$. Note that the number of vectors in $\bigcup_{h=0}^{d} \mathcal{P}_{hc}$ is at most $\sum_{h=0}^{d} p(hc)$. Thus the number of distinct subsets of $I$ exceeds the number of vectors in $\bigcup_{h=0}^{d} \mathcal{P}_{hc}$. By the pigeonhole principle, there are two distinct subsets $A$ and $B$ of $I$ such that $\sum_{i \in A} v_i = \sum_{i \in B} v_i$. Then $A' = A \setminus B$ and $B' = B \setminus A$ are two distinct and disjoint subsets of $I$ such that $\sum_{i \in A'} v_i = \sum_{i \in B'} v_i$.

Assume without loss of generality that $A' \neq \emptyset$.

Let $\lambda = \min\{ \lambda_i \mid i \in A' \}$. We can rewrite $u$ as follows:

$$\sum_{i \in I} \lambda_i v_i = \sum_{i \in I \setminus (A' \cup B')} \lambda_i v_i + \sum_{i \in A'} \lambda_i v_i + \sum_{i \in B'} \lambda_i v_i$$

$$= \sum_{i \in I \setminus (A' \cup B')} \lambda_i v_i + \sum_{i \in A'} (\lambda_i - \lambda) v_i + \sum_{i \in B'} (\lambda_i + \lambda) v_i = \sum_{i \in I} \mu_i v_i,$$

where

$$\mu_i = \begin{cases} 
\lambda_i & \text{for } i \in I \setminus (A' \cup B') \\
\lambda_i - \lambda & \text{for } i \in A' \\
\lambda_i + \lambda & \text{for } i \in B'.
\end{cases}$$

Note that $\mu_i \in \mathbb{Z}_{\geq 0}$ for all $i \in I$, and that $\mu_i = 0$ for at least one index $i \in A' \subseteq I$. Thus $u$ is expressed as the integer conical combination of at most $d - 1$ distinct vectors in $\mathcal{P}_c$.

By repeating the above argument, we can eventually obtain a representation of $u$ as the integer conical combination of at most $d(c)$ distinct vectors in $\mathcal{P}_c$. ⌣

If there is a multiset of vectors in $\mathcal{P}_c$ summing up to $\alpha$, then $\alpha$ is an integer conical combination of vectors in $\mathcal{P}_c$, and hence is included in $\text{int. cone}(\mathcal{P}_c)$, and hence by Lemma 3 can be represented as an integer conical combination of $d \leq d(c)$ distinct vectors in $\mathcal{P}_c$. Thus to decide whether there exists a multiset of vectors in $\mathcal{P}_c$ whose sum is $\alpha$, we can enumerate all subsets of $\{1, \ldots, p(c)\}$ of size $d \leq d(c)$, then for each subset $I$, check whether $\sum_{i \in I} \lambda_i v_i = \alpha$ has a solution with $\lambda_i \in \mathbb{Z}_{\geq 0}$ for all $i \in I$.

The checking step reduces to an integer linear program, with $O(c)$ linear constraints on at most $d(c) = O(c)$ variables $\lambda_i$, and with integer coefficients at most $kc$, where $\log kc = O(N)$.

We now analyze the running time of our algorithm. Recall that converting the item weights $w_i$ to multiplicities $a_w$ takes $N^{O(1)}$ time. Finding the $p(c)$ vectors in $\mathcal{P}_c$, $v_i$ for $1 \leq i \leq p(c)$, takes $c^{O(c)}$ time. The number of subsets of $\{1, \ldots, p(c)\}$ of size $d \leq d(c)$ is at most $(p(c))^{d(c)} = (2^{O(\sqrt{c})})^{O(c)} = 2^{O(c^{3/2})}$.

Solving an integer linear program with $n$ variables and $m$ constraints and with maximum coefficient $\Delta$ takes $n^{O(n)} m^{O(1)} (\log \Delta)^{O(1)}$ time\(^1\) [20, 18]. In our setting, this is $(O(c))^{O(c)} (O(c))^{O(1)} (O(N))^{O(1)} = c^{O(c)} N^{O(1)}$.

Thus the total running time is $N^{O(1)} + c^{O(c)} + 2^{O(c^{3/2})} \cdot c^{O(c)} N^{O(1)}$, which simplifies to $2^{O(c^{3/2})} N^{O(1)}$, since $c^{O(c)} = 2^{O(c \log c)}$. This completes the proof of Theorem 5.

\(^1\)See discussion at https://cstheory.stackexchange.com/questions/16530 for more refined estimates.
4 Dual approximation algorithm for bin packing

In this section we prove Theorem 6.

Fix any $0 < \epsilon \leq 1$. Separate the $n$ items into two groups: small items of weight at most $\epsilon c$, and large items of weight greater than $\epsilon c$.

Let $c' = \lceil 2\epsilon^{-2} \rceil$. Note that $c' \cdot \epsilon^2 c / 2 \leq c$. For each large item of weight $w_i > \epsilon c$, let $w'_i$ be the largest integer such that $w'_i \cdot \epsilon^2 c / 2 \leq w_i$. Then $(w'_i + 1) \cdot \epsilon^2 c / 2 > w_i$.

The algorithm works as follows. If the total weight of the $n$ items exceeds $kc$, then clearly the $n$ items cannot be packed into $k$ bins of capacity $c$, so return no. Otherwise, construct a set of scaled items including one item of weight $w'_i$ for each large item of weight $w_i$. Use the fixed-parameter algorithm in Theorem 5 to decide whether the set of at most $n$ scaled items $w'_i$ can be packed into $k$ scaled bins of capacity $c'$, then return the answer accordingly.

**Lemma 4.** If the scaled items can be packed into $k$ scaled bins of capacity $c'$, then the $n$ items can be packed into $k$ enlarged bins of capacity $(1 + \epsilon)c$.

**Proof.** For each large item of weight $w_i > \epsilon c$, the corresponding scaled item has weight

$$w'_i > \frac{w_i}{\epsilon^2 c / 2} - 1 > \frac{\epsilon c}{\epsilon^2 c / 2} - 1 = 2\epsilon^{-1} - 1 \geq \epsilon^{-1}.$$ 

Thus each scaled bin of capacity $c' = \lceil 2\epsilon^{-2} \rceil$ can hold at most $\lceil 2\epsilon^{-2} \rceil / \epsilon^{-1} \leq 2\epsilon^{-1}$ scaled items.

For the scaled items in each scaled bin, the sum of their weights $w'_i$ is at most $c'$, and hence the sum of $w'_i \cdot \epsilon^2 c / 2$ is at most $c' \cdot \epsilon^2 c / 2 \leq c$. Recall that $w'_i \cdot \epsilon^2 c / 2 \leq w_i < (w'_i + 1) \cdot \epsilon^2 c / 2$, and hence $w_i - w'_i \cdot \epsilon^2 c / 2 < \epsilon^2 c / 2$. Corresponding to the at most $2\epsilon^{-1}$ scaled items in each scaled bin, the total weight of the large items can exceed $c$ by at most $2\epsilon^{-1} \cdot \epsilon^2 c / 2 = \epsilon c$, and hence is at most $(1 + \epsilon)c$. Thus we can pack all large items into $k$ enlarged bins of capacity $(1 + \epsilon)c$.

Since the total weight of both large and small items is at most $kc$, at least one of the $k$ enlarged bins is filled to at most $c$ of its capacity $(1 + \epsilon)c$, and hence can always accommodate one more small item of weight at most $\epsilon c$. Thus we can pack all small items into the $k$ enlarged bins after the large items.

**Lemma 5.** If the scaled items cannot be packed into $k$ scaled bins of capacity $c'$, then the $n$ items cannot be packed into $k$ bins of capacity $c$.

**Proof.** We prove the contrapositive. Suppose that the $n$ items of total weight at most $kc$ can be packed into $k$ bins of capacity $c$. Then by discarding the small items and rounding down the weights $w_i$ of the large items to integer multiples of $\epsilon^2 c / 2$, the rounded items of weights $w'_i \cdot \epsilon^2 c / 2$ can be packed into $k$ bins of capacity $c = 2\epsilon^{-2} \cdot \epsilon^2 c / 2$, and hence the scaled items of integer weights $w'_i$ can be packed into $k$ scaled bins of capacity $c' = \lceil 2\epsilon^{-2} \rceil$.

With at most $n$ scaled items and capacity $c' = O(\epsilon^{-2})$ for the $k$ scaled bins, it follows by Theorem 5 that the running time of our dual approximation algorithm is $2^O(\epsilon^{-3}) N^{O(1)}$, where $N$ is the size of the problem instance encoded in binary. This completes the proof of Theorem 6.

5 Parameterized complexity of decomposing trees

In this section we prove Theorem 2. The problem $(k, c)$-DECOMPOSITION is clearly in NP. In the following, we prove that when both $k$ and $c$ are part of the input, $(k, c)$-DECOMPOSITION in trees is not only NP-hard but also W[1]-hard with parameter $k$, and is FPT with parameter $c$. 

9
5.1 NP-hardness and W[1]-hardness with parameter $k$

We give a polynomial reduction from Unary Bin Packing with $k$ bins to $(k, c)$-Decomposition in trees. Recall that Unary Bin Packing is W[1]-hard with parameter $k$ [16].

Given $n$ items of weights $w_i$, $1 \leq i \leq n$, and $k$ bins of capacity $c$, we construct a tree $T$ with $n(k - 1)c + \sum_{i=1}^{n} w_i$ edges as follows. For each item of weight $w_i$, make a star with $(k - 1)c + w_i$ edges. Then select an arbitrary leaf from each star, and merge the $n$ leaves selected from the $n$ stars into one center vertex, denoted by $v$.

We claim that the $n$ items can be packed into $k$ bins of capacity $c$ if and only if the tree $T$ can be decomposed into $k$ subgraphs with component size at most $c$.

We first prove the direct implication of the claim. Suppose that the $n$ items can be packed into $k$ bins of capacity $c$. We color the edges of the tree $T$ with $k$ colors as follows. First color the $n$ edges incident to the center vertex $v$ with $k$ colors corresponding to the $n$ items in the $k$ bins. Then for each star with $(k - 1)c + w_i$ edges, color $w_i - 1$ additional edges with the same color as the edge incident to the center vertex $v$, and color the remaining $(k - 1)c$ edges with the other $k - 1$ colors, $c$ edges of each color. Then the edges of each color induce a forest with component size at most $c$.

We next prove the reverse implication of the claim. Suppose that the tree $T$ can be decomposed into $k$ subgraphs with component size at most $c$. Color the edges of $T$ with $k$ colors corresponding to the $k$ subgraphs. Then in the star with $(k - 1)c + w_i$ edges corresponding to the item of weight $w_i$, the edge incident to the center vertex $v$ must have the same color as at least $w_i - 1$ other edges, because the other $k - 1$ colors can accommodate at most $(k - 1)c$ edges. Note that for each of the $k$ colors, the edges of this color that are incident to $v$, as well as the other edges of the same color in their respective stars, are all connected, and form one component of size at most $c$. Put the $n$ items into $k$ bins corresponding to the colors of the $n$ edges incident to $v$. Then the items in each bin must have total weight at most $c$ too.

5.2 FPT with parameter $c$

We present a fixed-parameter algorithm for $(k, c)$-Decomposition in trees with parameter $c$. Let $T$ be a tree with $n$ vertices and $n - 1$ edges, rooted at an arbitrary vertex of degree 1.

For each edge $e$ between a vertex $v$ and its parent in $T$, denote by $T_e$ the subtree of $T$ rooted at $v$ plus the edge $e$, and denote by $s(e)$ the minimum $s \geq 1$ such that $T_e$ admits an edge partition into $k$ subgraphs with component size at most $s$, under the constraint that $e$ is in a component of size at most $s$. If no such $s$ exists, let $s(e) = c + 1$. Then $T$ admits an edge partition into $k$ subgraphs with component size at most $c$ if and only if $s(e) \in [1, c]$ for all edges $e$.

We can compute $s(e)$ by dynamic programming, following a post-order traversal of the rooted tree $T$. For each edge $e$ incident to a leaf, we clearly have $s(e) = 1$. For each edge $e$ between an internal node $v$ and its parent, and for a candidate size $s \geq 1$, we can check whether $s(e) \leq s$ as follows.

Let $m_v$ be the number of edges incident to $v$. Construct $m_v$ items corresponding to the $m_v$ edges. Each edge $f$ between $v$ and a child corresponds to an item of weight $s(f)$, which has been computed following the post-order traversal. The edge $e$ between $v$ and its parent corresponds to an item of weight $c - s + 1$, where the surplus of $c - s$ in addition to 1 for the edge $e$ accounts for the difference between the desired component size $s$ for $e$ and the upper bound $c$. Then $T_e$ admits an edge partition into $k$ subgraphs with component size at most $c$, where $e$ is in a component of size at most $s$, if and only if the $m_v$ items can be packed into $k$ bins of capacity $c$.

By Theorem 5, there is a fixed-parameter algorithm for Bin Packing that decides whether the $m_v$ items thus constructed can be packed into $k$ bins of capacity $c$ in $2^{O(c^{3/2})}N_v^{O(1)}$ time, where $N_v = O(m_v(1 + \log kc))$ is the encoding size of the Bin Packing instance with $m_v$ items. By one invocation of this fixed-parameter algorithm with $s = c$, we either find that $s(e) > c$ and abort the traversal of $T$, or find that
In the latter case, we can then determine the exact value of $s(e)$ in $[1, c]$ by a binary search, using the fixed-parameter algorithm as a decision procedure.

The total running time of the algorithm is

$$n^{O(1)} + \sum_v \left( O(1 + \log c) \cdot 2^{O(c^{3/2})} N_v^{O(1)} \right) = 2^{O(c^{3/2})} n^{O(1)}.$$ 

Thus $(k, c)$-DECOMPOSITION in trees is FPT with parameter $c$. This completes the proof of Theorem 2.

### 6 Approximation algorithms for decomposing trees

In this section we prove Theorem 3 and Theorem 4. Let $T$ be a tree with $n$ vertices, $n - 1$ edges, and maximum degree $\Delta < n$, rooted at an arbitrary vertex of degree 1.

#### 6.1 Approximating MINIMUM-COLOR DECOMPOSITION

We first present a linear-time approximation algorithm for MINIMUM-COLOR DECOMPOSITION.

Perform a pre-order traversal of the rooted tree $T$. The root is incident to only one edge, which is colored arbitrarily. At each vertex other than the root, color the at most $\Delta - 1$ edges to its children with at most $\left\lceil \frac{\Delta - 1}{c} \right\rceil$ colors (at most $c$ edges of each color), all different from the color of the edge from its parent.

It is straightforward to verify that the edges of each color induce a forest with component size at most $c$, and moreover each component is a star with at most $c$ edges.

Let $k' = \left\lceil \frac{\Delta}{c} \right\rceil$ and $k'' = \left\lceil \frac{\Delta - 1}{c} \right\rceil + 1$. The minimum number of subgraphs with component size at most $c$ into which $T$ can be decomposed is at least $k'$. On the other hand, our algorithm colors the edges of $T$ with $k''$ colors, where $k'' \leq k' + 1$. This completes the proof of Theorem 3.

#### 6.2 Approximating MINIMUM-SIZE DECOMPOSITION

We next present a polynomial-time approximation scheme for MINIMUM-SIZE DECOMPOSITION.

We first note that our approximation algorithm for MINIMUM-COLOR DECOMPOSITION can be slightly modified to yield a 2-approximation for MINIMUM-SIZE DECOMPOSITION as follows. During the pre-order traversal of the rooted tree $T$, at each vertex other than the root, color the at most $\Delta - 1$ edges to its children with at most $k - 1$ colors (at most $\left\lceil \frac{\Delta - 1}{k} \right\rceil$ edges of each color), all different from the color of the edge from its parent. Let $c' = \left\lceil \frac{\Delta}{k} \right\rceil$ and $c'' = \left\lceil \frac{\Delta - 1}{k} \right\rceil$. In any edge partition of $T$ into $k$ subgraphs, there is a subgraph with component size at least $c'$. On the other hand, our algorithm colors the edges of $T$ with $k$ colors such that the edges of each color induce a forest with component size at most $c''$, where

$$c'' = \left\lceil \frac{\Delta - 1}{k - 1} \right\rceil = \left\lfloor \frac{k}{k - 1} \cdot \frac{\Delta - 1}{k} \right\rfloor \leq \left\lfloor \frac{k}{k - 1} \right\rfloor \cdot \left\lfloor \frac{\Delta - 1}{k} \right\rfloor \leq 2 \left\lfloor \frac{\Delta}{k} \right\rfloor = 2c'.$$

Now fix any $0 < \epsilon \leq 1$. To obtain a $(1 + \epsilon)$-approximation of the optimal component size $c^*$, it suffices to perform a binary search in the integer range $(c' - 1, c'')$, using a subroutine that decides either $c^* > c$ or $c^* \leq (1 + \epsilon)c$ for any candidate component size $c$. When the search range is reduced to $(l, h]$ with $h - l = 1$, we must have $c^* \geq h$, and the subroutine with $c = h$ decomposes the tree into $k$ subgraphs with component size at most $((1 + \epsilon)h)$.

The subroutine on a candidate component size $c$ is similar to our fixed-parameter algorithm with parameter $c$ in Theorem 2. As before, for each edge $e$ between a vertex $v$ and its parent in the rooted tree $T$, denote by $T_e$ the subtree of $T$ rooted at $v$ plus the edge $e$, and denote by $s(e)$ the minimum $s \geq 1$ such that $T_e$
admits an edge partition into \( k \) subgraphs with component size at most \( c \), under the constraint that \( e \) is in a component of size at most \( s \). If no such \( s \) exists, let \( s(e) = c + 1 \).

By definition, if \( s(e) \leq c \), then \( s(e) \) is the unique integer \( s \in [1, c] \) that satisfies the following two conditions:

A. \( T_e \) admits an edge partition into \( k \) subgraphs with component size at most \( c \), where \( e \) is in a component of size at most \( s \).

B. \( T_e \) does not admit any edge partition into \( k \) subgraphs with component size at most \( c \), where \( e \) is in a component of size at most \( s - 1 \).

Instead of computing \( s(e) \) directly, the subroutine tries to find a feasible value \( s \in [1, c] \) that satisfies the following two conditions, and records it in \( \tilde{s}(e) \):

C. \( T_e \) admits an edge partition into \( k \) subgraphs with component size at most \( (1 + \epsilon)c \), where \( e \) is in a component of size at most \( s \).

B. \( T_e \) does not admit any edge partition into \( k \) subgraphs with component size at most \( c \), where \( e \) is in a component of size at most \( s - 1 \).

The subroutine computes \( \tilde{s}(e) \) by dynamic programming, following a post-order traversal of the rooted tree \( T \). For each edge \( e \) incident to a leaf, set \( \tilde{s}(e) \leftarrow 1 \). For each edge \( e \) between an internal node \( v \) and its parent, check whether there exists a feasible value \( s \in [1, c] \) for \( \tilde{s}(e) \) that satisfies conditions C and B, as follows.

Let \( m_v \) be the number of edges incident to \( v \). Construct \( m_v \) items corresponding to the \( m_v \) edges. Each edge \( f \) between \( v \) and a child corresponds to an item of weight \( \tilde{s}(f) \). The edge \( e \) between \( v \) and its parent corresponds to an item of weight \( c - s + 1 \).

By Theorem 6, there is a dual approximation algorithm for \textsc{Bin Packing} that, given the \( m_v \) items and \( k \) bins of capacity \( c \), either decides that the \( m_v \) items can be packed into \( k \) enlarged bins of capacity \( (1 + \epsilon)c \) and returns yes, or decides that the \( m_v \) items cannot be packed into \( k \) bins of capacity \( c \) and returns no.

First call this dual approximation algorithm with \( s = c \). If its answer is no, then abort the traversal of \( T \), and declare that \( c^* > c \). Otherwise, find a feasible value \( s \in [1, c] \) for \( \tilde{s}(e) \) that satisfies both conditions C and B by a binary search, using the dual approximation algorithm as a decision procedure.

If the traversal of \( T \) finishes successfully, declare that \( c^* \leq (1 + \epsilon)c \).

The correctness of the subroutine can be verified by induction following the post-order traversal. In particular, for all edges \( e \) such that a feasible value \( s \in [1, c] \) is found for \( \tilde{s}(e) \), we have \( \tilde{s}(e) \leq s(e) \) because they satisfy the same condition B, while condition C is a relaxation of condition A. Also observe that the answers of the dual approximation algorithm have the following two implications:

- If the \( m_v \) items can be packed into \( k \) enlarged bins of capacity \( (1 + \epsilon)c \), then \( T_e \) admits an edge partition into \( k \) subgraphs with component size at most \( (1 + \epsilon)c \), where \( e \) is in a component of size at most \( s \).
- If the \( m_v \) items cannot be packed into \( k \) bins of capacity \( c \), then \( T_e \) does not admit any edge partition into \( k \) subgraphs with component size at most \( c \), where \( e \) is in a component of size at most \( s \).

By Theorem 6, the running time of the dual approximation algorithm on the \( m_v \) items is \( 2^{O(\epsilon^{-3})}N_v^{O(1)} \), where \( N_v = O(m_v(1 + \log kc)) \) is the encoding size of the problem instance. The total running time of the algorithm, including the nested binary searches on \( c^* \in (c' - 1, c'' \rangle \) and on \( s \in [1, c] \), is

\[
O(1) + O(\log(\Delta/k))\left( O(1) + \sum_v \left( O(1 + \log c) \cdot 2^{O(\epsilon^{-3})}N_v^{O(1)} \right) \right) = 2^{O(\epsilon^{-3})}n^{O(1)}.
\]

Thus we have a polynomial-time approximation scheme for \textsc{Minimum-Size Decomposition}. This completes the proof of Theorem 4.
7 Concluding remark

In our study of graph decomposition problems in this paper, the focus is on bipartite graphs and trees. It may be interesting to investigate the complexities of these problems on other important graph classes too.

References

[1] J. C. Bermond, J. L. Fouquet, M. Habib, and B. Peroche. On linear $k$-arboricity. *Discrete Mathematics*, 52:123–132, 1984.

[2] K. Bryš and Z. Lonc. Polynomial cases of graph decomposition: a complete solution of Holyer’s problem. *Discrete Mathematics*, 309:1294–1326, 2009.

[3] L. Cai and J. A. Ellis. NP-completeness of edge-colouring some restricted graphs. *Discrete Applied Mathematics*, 30:15–27, 1991.

[4] P. J. Cameron. *Combinatorics: Topics, Techniques, Algorithms*. Cambridge University Press, 1994.

[5] G. J. Chang, B.-L. Chen, H.-L. Fu, and K.-C. Huang. Linear $k$-arboricities on trees. *Discrete Applied Mathematics*, 103:281–287, 2000.

[6] R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21:5–12, 2001.

[7] D. Dor and M. Tarsi. Graph decomposition is NP-complete: a complete proof of Holyer’s conjecture. *SIAM Journal on Computing*, 26:1166–1187, 1997.

[8] F. Eisenbrand and G. Shmonin. Carathéodory bounds for integer cones. *Operations Research Letters*, 34:564–568, 2006.

[9] T. Emden-Weinert, S. Hougardy, and B. Kreuter. Uniquely colourable graphs and the hardness of colouring graphs of large girth. *Combinatorics, Probability and Computing*, 7:375–386, 1998.

[10] A. Farrugia. Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. *The Electronic Journal of Combinatorics*, 11:#R46, 2004.

[11] H. N. Gabow and H. H. Westermann. Forests, frames, and games: algorithms for matroid sums and applications. *Algorithmica*, 7:465–497, 1992.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, 1979.

[13] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[14] D. S. Hochbaum (editor). *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[15] I. Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10:718-720, 1981.

[16] K. Jansen, S. Kratsch, D. Marx, and I. Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79:39–49, 2013.
[17] M. Jiang. Trees, paths, stars, caterpillars and spiders. *Algorithmica*, 80:1964-1982, 2018.

[18] R. Kannan. Minkowski’s convex body theorem and integer programming. *Mathematics of Operations Research*, 12:415–440, 1987.

[19] J. Kleinberg, R. Motwani, P. Raghavan, and S. Venkatasubramanian. Storage management for evolving databases. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS’97)*, pages 353–362, 1997.

[20] H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

[21] D. Leven and Z. Galil. NP completeness of finding the chromatic index of regular graphs. *Journal of Algorithms*, 4:35–44, 1983.

[22] J. H. van Lint and R. M. Wilson. *A Course in Combinatorics*. Second Edition. Cambridge University Press, 2001.

[23] L. Lovász. Coverings and colorings of hypergraphs. In *Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 3–12, Utilitas Mathematica Publishing, Winnipeg, 1973.

[24] F. Maffray and M. Preissmann. On the NP-completeness of the $k$-colorability problem for triangle-free graphs. *Discrete Mathematics*, 162:313–317, 1996.

[25] J. Misra and D. Gries. A constructive proof of Vizing’s theorem. *Information Processing Letters*, 41:131–133, 1992.

[26] C. St.J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 39:12, 1964.