The Complexity of Packing Edge-Disjoint Paths

Jan Dreier
Dept. of Computer Science, RWTH Aachen University, Germany
dreier@cs.rwth-aachen.de

Janosch Fuchs
Dept. of Computer Science, RWTH Aachen University, Germany
fuchs@algo.rwth-aachen.de

Tim A. Hartmann
Dept. of Computer Science, RWTH Aachen University, Germany
hartmann@algo.rwth-aachen.de

Philipp Kuinke
Dept. of Computer Science, RWTH Aachen University, Germany
kuinke@cs.rwth-aachen.de

Peter Rossmanith
Dept. of Computer Science, RWTH Aachen University, Germany
rossmani@cs.rwth-aachen.de

Bjoern Tauer
Dept. of Computer Science, RWTH Aachen University, Germany
tauer@algo.rwth-aachen.de

Hung-Lung Wang
Computer Science and Information Engineering, National Taiwan Normal University, Taiwan
hlwang@gapps.ntnu.edu.tw

Abstract

We introduce and study the complexity of Path Packing. Given a graph G and a list of paths, the task is to embed the paths edge-disjoint in G. This generalizes the well known Hamiltonian-Path problem.

Since Hamiltonian Path is efficiently solvable for graphs of small treewidth, we study how this result translates to the much more general Path Packing. On the positive side, we give an FPT-algorithm on trees for the number of paths as parameter. Further, we give an XP-algorithm with the combined parameters maximal degree, number of connected components and number of nodes of degree at least three. Surprisingly the latter is an almost tight result by runtime and parameterization. We show an ETH lower bound almost matching our runtime. Moreover, if two of the three values are constant and one is unbounded the problem becomes NP-hard.

Further, we study restrictions to the given list of paths. On the positive side, we present an FPT-algorithm parameterized by the sum of the lengths of the paths. Packing paths of length two is polynomial time solvable, while packing paths of length three is NP-hard. Finally, even the spacial case Exact Path Packing where the paths have to cover every edge in G exactly once is already NP-hard for two paths on 4-regular graphs.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases parameterized complexity, embedding, packing, covering, Hamiltonian path, unary binpacking, path-perfect graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.10

Related Version A full version of the paper is available at `https://arxiv.org/abs/1910.00440`.

14th International Symposium on Parameterized and Exact Computation (IPEC 2019).
Editors: Bart M. P. Jansen and Jan Arne Telle; Article No. 10; pp. 10:1–10:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Packing, covering and partitioning are well researched fields in graph theory. In general, the task is to cover a given graph $G = (V, E)$ with or partition it into smaller substructures, or to pack given structures into the graph. Besides that these terms are often used, they are not well defined throughout the literature. Thus, it is important to define problems in this field carefully and in detail.

For example, the path partition problem is a well studied problem [2, 4, 6, 12, 18, 21, 26] which is also known as path cover problem. The task is to cover all vertices of a graph with vertex-disjoint paths. This is equivalent to partitioning the graph into vertex-disjoint paths. The smallest number of paths to achieve this is called the path partition number or path cover number. Observe that G has a Hamiltonian path iff the path-partition number is one, thus the problem is NP-complete.

An NP-complete variant of this problem is the k-path partition problem [22, 27, 19]. Here the task is to partition a graph G into paths, such that none of the path lengths exceeds k. Observe that the 1-path-partition problem corresponds to finding a maximum matching.

Another related problem is the recognition of path-perfect graphs [5, 11, 13, 23, 29], which we denote in this work as Path-Perfect Packing. Instead of partitioning the graph into vertex-disjoint paths, the complete edge set must be partitioned into edge-disjoint paths of ascending length, starting by one. This can also be understood as packing k paths of length 1 to k into G without using an edge twice or leaving one edge uncovered.

This approach of packing smaller subgraphs into a given graph is also well researched [28]. For example, packing edge-disjoint trees into a clique is considered [24]. Since packing edge-disjoint and vertex-disjoint triangles is NP-hard for planar graphs, the parameterized complexity is studied [7].

We generalize the path-perfect graph problem and ask for a given graph G and a list of k paths $P = \{p_1, \ldots, p_k\}$ if they can be embedded into G without using the same edge twice. Note that we define the length of a path equals its number of edges. This problem arises naturally when restricting the path partition problem to edge-disjoint paths instead of vertex-disjoint paths. We denote this problem as Path Packing. Let us formalize what we mean by embedding. An embedding of a graph H into a graph G is an injective mapping $f : V(H) \to V(G)$ such that for every original edge $(u, v) \in E(H)$ also $(f(u), f(v)) \in E(G)$. An embedding of a list of graphs $\mathcal{H}$ into G is an embedding of each graph H into G. Note, that we do not ask to embed the graphs pairwise vertex-disjointly. The embeddings we consider in this work are pairwise edge-disjoint embeddings of paths.

Path Packing
Input: A list of paths $P = \{p_1, \ldots, p_k\}$ of length $l_1, \ldots, l_k$. A graph $G = (V, E)$.
Question: Is there an edge-disjoint embedding of P into G?

The Exact Path Packing problem additionally requires that every edge is covered *exactly* once.

Exact Path Packing
Input: A list of paths $P = \{p_1, \ldots, p_k\}$ of length $l_1, \ldots, l_k$. A graph $G = (V, E)$.
Question: Is there an edge-disjoint embedding of P into G such that each edge $e \in E$ is covered exactly once?

Path Packing is clearly more general than Exact Path Packing, since one can reduce from one to the other by additionally requiring the sum of the path lengths to be equal to the number of edges in the graph. Most of our hardness results are for Exact Path Packing, and therefore translate to Path Packing. Our upper bounds are always regarding more the general Path Packing.

Our Results

The Hamiltonian path problem is a special case of Path Packing. An even though the Hamiltonian path problem is tractable on graphs of bounded treewidth, Path Packing is already NP-complete on subdivided stars. Therefore, we focus on the parameterized complexity to classify this problem on a finer scale. We will analyze the impact of various parameters.

In Section 3, we analyze the parameterized complexity of our packing problems with respect to the number of paths (denoted by k). On the one hand, we give an FPT algorithm for Path Packing that solves the problem in time $2^k n^{O(1)}$ on subcubic (i.e. degree at most three) forests (Theorem 3). On the other hand, we show that Exact Path Packing on graphs with treewidth two is $W[1]$-hard and cannot be solved in time $f(k)n^{o(k/\log k)}$ under ETH (Theorem 11).

In Section 4 we introduce path dependent restrictions. We show that Exact Path Packing is NP-complete even for two paths on 4-regular graphs (Theorem 14). length i is easy for $i = 2$ and NP-complete for $i = 3$ (Theorem 15). If we however parameterize by the summed length of all paths Path Packing is in FPT (Theorem 16).

After parameterizing by the number of paths and their lengths, we further analyze graph dependent parameters in Section 5. We introduce the *bcd-number* of a graph, which is the maximum of the number of components, the maximal degree, and the number of vertices with degree larger than two. We show that Path Packing can be solved in time $k!^k(n+k^2)^{O(k^2)}$, where k is the bcd-number (Theorem 20). This is complemented by showing that the problem cannot be solved in $f(k)n^{o(k^2/\log k)}$ under ETH (Theorem 21). We further show that all three bcd parameters are necessary: If two values are constant and one is unbounded the problem becomes NP-hard (Theorem 1, Corollary 18, Theorem 19).

Note that, one can embed paths $p_1, \ldots, p_k$ as edge-disjoint subgraphs into a graph G if and only if one can embed these paths as vertex-disjoint induced subgraphs into the linegraph of G. Therefore, our results yield new insights for the problem of covering a graph with a list of vertex-disjoint induced paths [17]. Especially, our hardness results for certain graph classes transfer to hardness results on the linegraphs of these graph classes. Due to space limitations, we omit some proofs, and refer to the full version.

2 Preliminaries

All graphs are simple (i.e. without multi-edges or self-loops). The length of a path equals its number of edges.

3 Path Packing on Forests

Our packing problem is a generalization of the Hamiltonian path problem and therefore NP-complete. The Hamiltonian path problem is solvable in polynomial time if the treewidth of the input graph is bounded [9]. We show that (unlike Hamiltonian path) Exact Path Packing is NP-complete on trees. This is done by reducing it to the following NP-complete partitioning problem.

Multi-Way Number Partition

Input: A list of weights $w_1, \ldots w_n \in \mathbf{N}$ encoded in unary, and an integer $k \in \mathbf{N}$.

Question: Is there a partition of $w_1, \ldots w_n$ into k multi-sets $S_1, \ldots, S_k$ such that $\sum_{w_i \in S_j} w_i = \frac{1}{k} \sum_{i=1}^{n} w_i$, for every $1 \leq j \leq k$?

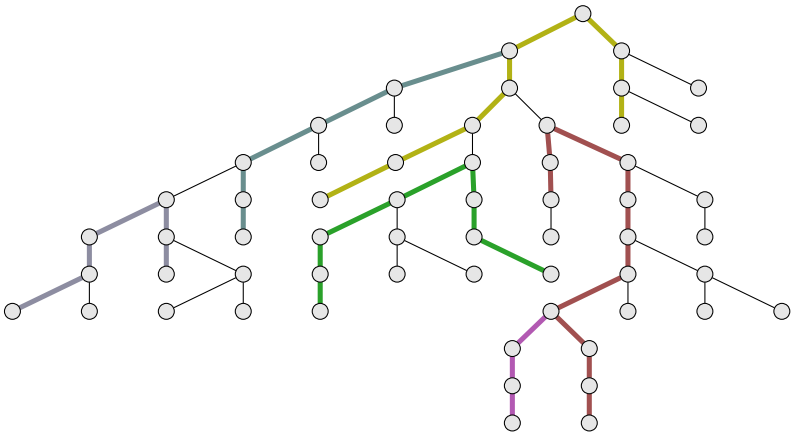

Figure 1 Packing paths of lengths 10, 8, 7, 5, 5, 3 into a subcubic tree. Although the packing looks very loose there is no solution if we replace 3 by 4.

We reduce from Multi-Way Number Partition to prove that Exact Path Packing is NP-hard on very simple trees.

▶ **Theorem 1.** Exact Path Packing *is* NP*-complete on subdivided stars.*

The previous reduction required a large number of paths. Therefore, in the following, we analyze the Path Packing problem parameterized by the number of paths.

Fast subset convolution

We develop dynamic programming algorithms on subcubic trees whose running time will be $O^*(2^k)$, where k is the number of paths that we want to pack. First we develop a naive and not too complicated algorithm with running time $O^*(3^k)$, whose longer running time is due to some very simple operation that occurs when we combine two dynamic programming tables. Björklund, Husfeldt, Kaski, and Koivisto introduced a technique called *fast subset convolution* that was used to speed up the computation of Steiner trees with small integer weights [3] and also to speed up some algorithms that do dynamic programming on tree decompositions [25]. We can use this technique to our advantage to significantly speed up the path packing algorithm on trees. The result that we will be using is:

▶ **Proposition 2.** [3] *Let N be a set of n natural numbers and $f, g\colon \mathbf{N} \to \mathbf{N}$ two functions. Then we can compute $(f * g)(S)$ for all $S \subseteq N$ in time $O(2^n n^2)$ if f and g can be evaluated in constant time and where $(f * g)(S) = \sum_{T \subset S} f(T)g(S - T)$. Here $f(S) = \sum_{i \in S} f(i)$.*

▶ **Theorem 3.** *We can solve* Path Packing *for k paths in time $O^*(2^k)$ on subcubic forests.*

Proof. Let us assume that the graph is a subcubic tree T, but the proof easily generalizes to subcubic forests. Let $l_1, \ldots, l_k$ be length of the paths that we want to pack into T. We can further assume that T is a rooted tree by designating an arbitrary vertex as its root. If v is a vertex of T then let $T(v)$ be the subtree rooted at v.

We solve the path packing problem by dynamic programming computing a table for each vertex in a bottom-up order. Such a table is a mapping $L\colon V \times 2^{[k]} \to [n] \cup \{-\infty\}$. The size of this table is $O(2^k n)$. We interpret the content of the table as follows:

$L(v, P) = r$ with $r \geq 0$ means that it is possible to pack all paths with indices in P (in short all P-paths) into the subtree $T(v)$ and additionally a path of length r that ends in v.

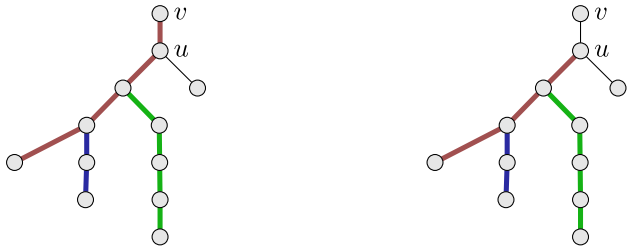

Figure 2 Left side: Packing paths of lengths $l_1 = 4$, $l_2 = 4$, $l_3 = 2$ into $T(v)$. $L(u, \{1, 2, 3\}) = -\infty$, but $L(u, \{1, 2, 3\} - \{1\}) = l_1 - 1$, so $L(v, \{1, 2, 3\}) = 0$.
Right side: Now $l_1 = 4$, $l_2 = 3$, $l_3 = 2$. $L(u, \{1, 2, 3\}) = 1$, so $L(v, \{1, 2, 3\}) = 1 + 1 = 2$. An additional path of length 2 can be packed into $T(v)$, because an additional path of length 1 can be packed into $T(u)$.

The special case $L(v, P) = 0$ means that we can pack all P-paths into $T(v)$, but however we pack them there is no space left to pack another path that ends in v.

If it is not possible to pack all P-paths into $T(v)$ at all then let $L(v, P) = -\infty$.

It is quite clear that having computed all tables enables us to find out whether (T, P) is a yes-instance of the path packing-problem. Simply check whether $L(r, \{1, \dots, k\}) \neq -\infty$.

To compute the tables for all v we distinguish three cases how to compose trees into bigger trees: **1** v is a leaf, **2** v has one child, **3** v has at least two children.

Leaf. If v is a leaf then $L(v, \emptyset) = 0$ and $L(v, P) = -\infty$ if $P \neq \emptyset$ because we cannot pack any path into an empty tree (that has no edges).

One child. If v has one child u then it is also quite easy to compute $L(v, P)$: If $L(u, P) = r$ with $r \geq 0$, then clearly $L(v, P) = r + 1$. The right hand side of Figure 2 shows an example. The more complicated possibility is $L(u, P) = -\infty$, which means that it is completely impossible to pack all P-paths into $T(u)$. It might become possible to pack all P-paths into $T(v)$ by using the additional edge uv. If this is possible, then one path, say the ith one with length l_i, uses the edge uv. Then all paths in $P - \{i\}$ are packed into $T(u)$ and one additional path of length $l_i - 1$ that ends in u. We can check this by verifying that $L(u, P - \{i\}) \geq l_i - 1$ for some $1 \leq i \leq k$ (actually, $L(u, P - \{i\}) \geq l_i$ is impossible, because then all P-paths could be packed into $T(u)$ and $L(u, P) \neq -\infty$). If we find such an i, then all P-paths can be packed into $T(v)$, but only by using the edge uv. This means that no other path can be packed into $T(v)$ that ends in v and therefore $L(v, P) = 0$. See the left hand side of Figure 2 for an example.

$$L(v, P) = \begin{cases} L(u, P) + 1 & \text{if } L(u, P) \geq 0 \\ 0 & \text{if } L(u, P) = -\infty \text{ and } L(u, P - \{i\}) = l_i - 1 \text{ for some } i \in P \\ -\infty & \text{otherwise} \end{cases}$$

Two children. Finally, we assume that v has exactly two children u_1, u_2. In that case we can construct for each of them a new tree by attaching new roots v_1, v_2 to $T(u_1)$ and $T(u_2)$ and computing the L-tables for both of them. To compute the table of v it is sufficient to compute a table for a tree that we get by glueing two trees together by identifying their roots. We just have to glue v_1 to v_2.

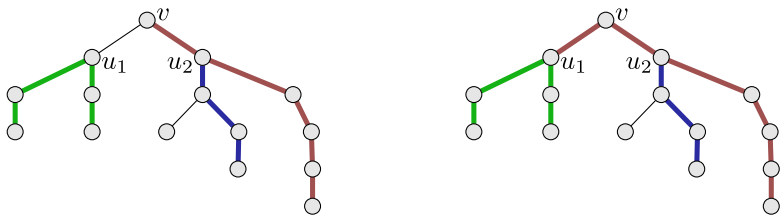

Figure 3 Left side: Packing paths of lengths $l_1 = 5$, $l_2 = 4$, $l_3 = 3$ into $T(v)$. $L(v_1, \{2\}) = 1$ and $L(v_2, \{1, 3\}) = 0$ imply $L(v, \{1, 2, 3\}) \geq 1$.
Right side: Now $l_1 = 6$, $l_2 = 4$, $l_3 = 3$. $L(v_1, \{2\}) + L(v_2, \{3\}) = 1 + 5 = l_1$ implies $L(v, \{1, 2, 3\}) \geq 0$. This time we partition $\{1, 2, 3\}$ into three parts. One goes to the left, one to the right, and one path uses both subtrees.

So we can assume that we have two trees with roots v_1 and v_2 and a tree with root v that we get by identifying v_1 and v_2 and renaming it to v. This is often called a join operation. We have the tables for v_1 and v_2 and want to compute the table for v.

Clearly, $L(v, P) = r$ with $r > 0$ iff some of the P-paths can be packed into $T(v_1)$ and the others into $T(v_2)$ and the additional path with length r that ends in v can be packed into $T(v_1)$ or $T(v_2)$. The additional path of length r that ends in v prevents any P-path from being packed partially into $T(v_1)$ and $T(v_2)$. That is the case iff there is a bipartition of P into P_1 and P_2 such that $L(v_1, P_1), L(v_2, P_2) \geq 0$ and $\max\{L(v_1, P_1), L(v_2, P_2)\} = r$. There are $2^{|P|}$ many subsets of P. To check all bipartitions for all these subsets $P \subseteq [k]$ means looking at $\sum_{i=0}^{k} \binom{k}{i} 2^i = 3^k$ many cases. Using fast subset convolution lets us speed up the computation to $2^k k^{O(1)}$ steps: Let

$$f_i(S) = \begin{cases} 1 & \text{if } L(v_i, S) \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad g_i(S) = \begin{cases} 1 & \text{if } L(v_i, S) \geq r \\ 0 & \text{otherwise.} \end{cases}$$

Then $L(v, P) \geq r$ iff $(f_1 * g_2)(P) + (g_1 * f_2)(P) \geq 1$.

The situation is different if $L(v, P) = 0$. In that case both edges u_1v and u_2v have to be used when packing all P-paths into $T(v)$ because otherwise at least a path of length one that ends in v could additionally be packed into $T(v)$.

In such a packing one path, say the ith one with length l_i, uses u_1v and u_2v. That is possible iff there is a bipartition of $P - \{i\}$ into P_1 and P_2 such that $L(v_1, P_1) + L(v_2, P_2) \geq l_i$. Again, by using fast subset convolution we can check this in $2^k k^{O(1)}$ steps. ◀

The above proof does not work any more if we glue together two trees whose roots have degree higher than one. For general trees the dynamic programming is much more complicated and we will need more complicated tables.

Let $T(v)$ be a rooted tree with root v and no restrictions on the degree of vertices (and thus on the number of children). Let us again fix length $l_1, \ldots, l_k \in \mathbf{N}$ of paths that we are going to pack into a tree. We are going to identify a set of paths by a set $P \subseteq [k]$. We speak of P-paths as the paths with length l_i for every $i \in P$.

▶ **Definition 4.** Let us fix $l_1, \ldots, l_k \in \mathbf{N}$, $P \subseteq [k]$, and T be a rooted tree. $T(v)$ is the subtree of T with root v.

1. Let $M, M' \subseteq \mathbf{N}$ be two multisets. We say that $M' \succcurlyeq M$ if we can construct M' from M by adding numbers and increasing numbers that are already in M'.
   Let $M' \succ M$ iff $M' \neq M$ and $M' \succcurlyeq M$.
   Example: $\{3, 3, 5, 5, 7\} \succcurlyeq \{2, 3, 4, 6\}$, but $\{3, 3, 5, 5, 7\} \not\succcurlyeq \{2, 3, 4, 8\}$.

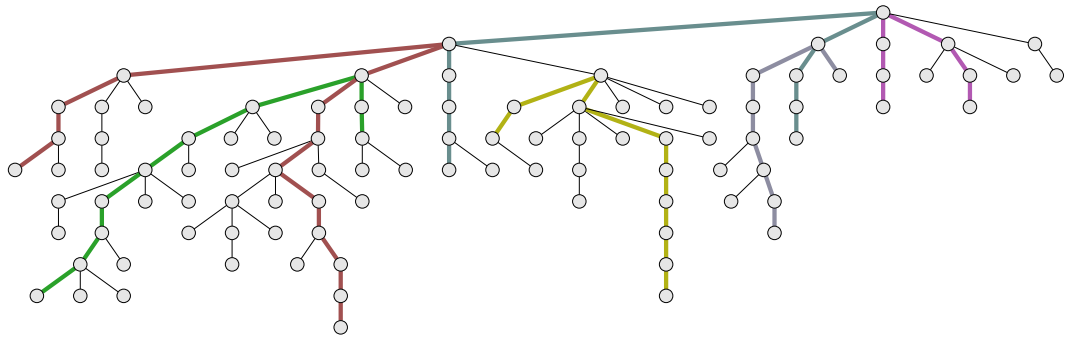

Figure 4 In this tree there are nodes with more than two children and paths can "cross." We pack paths with lengths $13, 9, 9, 9, 7, 6$. There is no solution if we replace 6 by 7.

2. Let $\mathcal{S}$ be a set of multisets of natural numbers. Then

$$K(\mathcal{S}) = \{\, M \in \mathcal{S} \mid \text{ there is no } M' \in \mathcal{S} \text{ with } M' \succ M \,\}.$$

3. Then we define $\mathcal{L}(v, P)$ as a set of multisets of natural numbers as follows:
Let $M \subseteq \mathbf{N}$ be a multiset of natural numbers. Then $M \in \mathcal{L}(v, P)$ iff it is possible to pack all P-paths into $T(v)$ such that we can pack additionally all non-empty paths into $T(v)$ that start at v and have lengths given in M and if there is no $M' \in \mathcal{L}(v, P)$ with $M' \succcurlyeq M$.
Particularly, $\mathcal{L}(v, P) = \emptyset$ iff it is impossible to pack all P-paths into $T(v)$ and $\mathcal{L}(v, P) = \{\emptyset\}$ iff it is possible to pack all P-paths into $T(v)$, but there is no possibility to additionally pack a non-empty path that starts at v.
4. If $M \subseteq \mathbf{N}$ then $\max_q(M)$ is the multiset that consists of the q biggest elements in M or of all of them if M contains less than q numbers, e.g., $\max_3(\{5, 5, 4, 4, 3, 2, 1\}) = \{5, 5, 4\}$.

In the following let $l_1, \ldots, l_k$ be fixed.

▶ **Lemma 5.** *Let $T(v)$ be a rooted tree with root v such that v has one child u.*

1. *Assume that $\mathcal{L}(u, P) = \{M_1, \ldots, M_m\}$ with $m \geq 1$. Define $\mathcal{L}_{\max}(u, P) = \max(M_1 \cup \cdots \cup M_m)$ (where $\max \emptyset = 0$).*
Then $\mathcal{L}(v, P) = \{\{\mathcal{L}_{\max}(u, P) + 1\}\}$.
2. *Assume that $\mathcal{L}(u, P) = \emptyset$ and there is an $i \in \{1, \ldots, k\}$ with $\mathcal{L}_{\max}(u, P - \{i\}) = l_i - 1$.*
Then $\mathcal{L}(v, P) = \{\emptyset\}$.
3. *Otherwise $\mathcal{L}(v, P) = \emptyset$.*

Proof. We have to consider exactly two cases. The first case is that it is possible to pack all P-paths into $T(u)$. If this is the case, then an additional path of length $r + 1$ can be packed into $T(v)$ starting at v iff an additional path of length r can be packed into $T(u)$ starting at u. The latter is the case iff $\mathcal{L}_{\max}(u, P) = r$.

The second case is that it is impossible to pack all P-paths into $T(u)$ alone. It might still be possible to pack them into $T(v)$, but only if the edge uv is used. This means that there is only space for an additional path of length zero that starts at v.

In fact, exactly one path, say the ith one, uses the edge uv. This is possible iff we can pack all $(P - \{i\})$-paths into $T(u)$ and being able to additionally pack a path of length at least $l_i - 1$ into $T(u)$ starting at u. Actually, this path cannot be longer than $l_i - 1$ because then we would be able to pack all P-paths, which is a contradiction. ◀

▶ **Lemma 6.** *Let $T(v_1)$ and $T(v_2)$ be two rooted trees with no common vertices, such that v_2 has exactly one child. Let $T(v)$ be the tree that we get by identifying v_1 with v_2 and renaming it to v. Then $\mathcal{L}(v,P) = K(\mathcal{L}_1 \cup \mathcal{L}_2)$ where*

$$\mathcal{L}_1 = \bigcup_{\substack{P_1 \subseteq P \\ P_2 = P - P_1}} \bigcup_{\substack{M_1 \in \mathcal{L}(v_1, P_1) \\ M_2 \in \mathcal{L}(v_2, P_2)}} \{M_1 \cup M_2\}$$

$$\mathcal{L}_2 = \bigcup_{\substack{P_1 \subseteq P \\ P_2 = P - P_1}} \bigcup_{i \in P} \bigcup_{\substack{M_1 \in \mathcal{L}(v_1, P_1 - \{i\}) \\ M_2 \in \mathcal{L}(v_2, P_2 - \{i\})}} \bigcup_{\substack{r_1 \in M_1 \\ r_2 \in M_2 \\ r_1 + r_2 \geq l_i}} \{(M_1 - \{r_1\}) \cup (M_2 - \{r_2\})\}$$

Proof. "$\mathcal{L}(v,P) \supseteq K(\mathcal{L}_1 \cup \mathcal{L}_2)$": If $M \in \mathcal{L}_1 \cup \mathcal{L}_2$ then $M \in \mathcal{L}_1$ or $M \in \mathcal{L}_2$. Let us first consider the case $M \in \mathcal{L}_1$. By the definition of $\mathcal{L}_1$ there are $P_1 \subseteq P$, $P_2 = P - P_1$, $M_1 \in \mathcal{L}(v_1, P_1)$, and $M_2 \in \mathcal{L}(v_2, P_2)$ such that $M = M_1 \cup M_2$. By induction we know that P_1 can be packed into $T(v_1)$ as well as additional paths of lengths M_1 starting at v_1. The same holds for P_2, v_2, and M_2. Using this packing we actually packed P into $T(v)$ and additional paths of lengths $M_1 \cup M_2 = M$ starting at v. By definition then $M \in \mathcal{L}(v,P)$.

The other possibility is $M \in \mathcal{L}_2$, which is a bit more complicated. If $M \in \mathcal{L}_2$, then $M = (M_1 - \{r_1\}) \cup (M_2 - \{r_2\})$, where $r_1 \in M_1$, $r_2 \in M_2$, $r_1 + r_2 \geq l_i$, $M_1 \in \mathcal{L}(v_1, P_1 - \{i\})$, $M_2 \in \mathcal{L}(v_2, P_2 - \{i\})$, $P_1 \subseteq P$, $P_2 = P - P_1$, and $i \in P$.

We have to show that it is possible to pack P into $T(v)$ and additionally paths with lengths from M starting at v. By induction we know that we can pack all $(P_1 - \{i\})$-paths into $T(v_1)$ and all $(P_2 - \{i\})$-paths into $T(v_2)$. Simultaneously, we can pack additional paths with lengths from M_1 into $T(v_1)$ starting at v_1 and paths with lengths from M_2 into $T(v_2)$ starting at v_2. Hence, we can pack paths with lengths $M = (M_1 - \{r_1\}) \cup (M_2 - \{r_2\})$ into $T(v)$ leaving space for a path of length r_1 in $T(v_1)$ and a path of length r_2 in $T(v_2)$. We can combine these two paths into one path of length $r_1 + r_2 \geq l_i$ and pack one additional path of length l_i into $T(v)$. Altogether we packed P_1, P_2, $\{i\}$ and therefore all P-paths into $T(v)$.

"$\mathcal{L}(v,P) \subseteq K(\mathcal{L}_1 \cup \mathcal{L}_2)$": Let $M \in \mathcal{L}(v,P)$. Then P can be packed into $T(v)$. There are two possibilities:

1. No path corresponding to $i \in P$ lies partially in $T(v_1)$ and partially in $T(v_2)$. Then we can split $P = P_1 \cup P_2$ such that P_1-paths are packed into $T(v_1)$ and P_2-paths into $T(v_2)$. The additional path with lengths from M are also packed into $T(v_1)$ and $T(v_2)$. Let us say $M = M_1 \cup M_2$, where M_1 is in $T(v_1)$ and M_2 in $T(v_2)$. Then it is easy to see that $M \in \mathcal{L}_1$.

2. There is an $i \in P$ such that all $(P - \{i\})$-paths are packed into $T(v_1)$ and $T(v_2)$, but exactly one path with length l_i is packed into $T(v)$ using edges from both $T(v_1)$ and $T(v_2)$. Note that there can be at most one such path because v_2 has only one child in $T(v_2)$. Then all additional paths with lengths in M that start at v have to be packed into $T(v_1)$ alone because the edge in $T(v_2)$ is not available any more. Let r_1 be the length of the part of the bridging path of length l_i that lies in $T(v_1)$ and r_2 the length of the part in $T(v_2)$. Clearly, $r_1 + r_2 = l_i$. With all these facts we can again easily verify that $M \in \mathcal{L}_2$. ◀

The following lemma shows that the size of the tables is bounded by a function in k and the maximal degree. The estimate is quite pessimistic, but we are not trying to optimize the runtime of the dynamic programming algorithm at the moment and are content with proving fixed parameter tractability.

▶ **Lemma 7.** *Let $T(v)$ be a rooted tree and assume that vertex v has d children. Then*

$$|\mathcal{L}(v,P)| \leq d2^{kd}.$$

Proof. If v has only one child, then $|\mathcal{L}(v,P)| = 1$ and the statement is true. Assume next that $T(v)$ has d children. Each subtree can receive at most 2^k different sets of packed paths yielding at most 2^k different length of the longest path that can be additionally packed. Therefore a set $M \in \mathcal{L}(v,P)$ can have size at most d and contain up to d numbers each chosen from a set of size at most 2^k. In total that are at most $d2^{kd}$ possibilities for a set M. ◀

▶ **Theorem 8.** *Let T be a rooted tree and P a multiset of paths. In polynomial time a rooted tree T' can be computed that has the following properties:*
(1) *P can be packed into T iff it can be packed into T',*
(2) *each node in T' has at most $3|P|$ children, and (3) T' is a subtree of T.*

Proof. Let $l_1(u)$ be the length of the longest path in $T(u)$ that starts in u and $l_2(u)$ be the length of the longest path in $T(u)$. Assume that P can be packed into T and v be an arbitrary vertex in T. Let us fix an edge-disjoint packing of P.

Let v be an arbitrary node in T and $v_1, \ldots, v_m$ the children of v. Let us further assume that $v_1, \ldots, v_{3|P|}$ contain the $|P|$ children with biggest $l_1(v_i)$ and $2|P|$ children with biggest $l_2(v_i)$. Ties can be arbitrarily ordered.

If $m \leq 3|P|$ we do nothing. Otherwise assume that P is packed into T and some path $p \in P$ uses $T(v_i)$ with $i > 3|P|$. There are two possibilities:

(i) p contains v_i. Then p is possibly packed partially inside $T(v_i)$ and partially outside. Let p' be the part of p inside $T(v_i)$. Clearly, p' starts at v_i. By the pigeonhole principle there must be some $T(v_k)$ that has not been used in the packing of P, $l_1(v_k) \geq l_1(v_i)$, and $k \leq 3|P|$. Then we can repack p such that it uses $T(v_k)$ instead of $T(v_i)$.

(ii) p does not contain v_i and is therefore completely packed into $T(v_i)$. Again by the pigeonhole principle we can find an appropriate $T(v_k)$ with $k \leq 3|P|$ and $l_2(v_k) \geq l_2(v_i)$. We can repack p from $T(v_i)$ into $T(v_k)$.

Repeated repacking in these two ways leads to a packing that uses only the subtrees $T(v_1), \ldots, T(v_{3|P|})$. We can therefore remove all other subtrees without changing a yes-instance into a no-instance. Applying this pruning to all vertices in T leads to a new tree T' that has all properties stated in the theorem. It is also clear that T' can be computed in polynomial time as it is easy to find longest paths in trees. ◀

Combining the above results (with the base case for a leaf v: $\mathcal{L}(v,P) = \{\{0\}\}$ if P contains only empty paths and $\mathcal{L}(v,P) = \emptyset$ otherwise) we can prove the following:

▶ **Theorem 9.** Path Packing *into forests parameterized by the number of paths is in* FPT.

Proof. Given a tree T compute a rooted tree T' where each node has at most $3k$ children and every P (with $|P| = k$) can be packed into T iff it can be packed into T' (Theorem 8). Then use dynamic programming to find out whether the paths can be packed into T'. By Lemma 7 and 6 this only takes time $f(k)|T|^{O(1)}$ for some function f. ◀

Lower bound

While Path Packing on graphs with treewidth one is in FPT when parameterized by the number of paths, we now show that the problem becomes hard on graphs with treewidth two. As an intermediate step, we reduce from Unary Bin Packing [15] to show hardness of Multi-Way Number Partition. This then leads to hardness results for Exact Path Packing. Remember that for Multi-Way Number Partition the numbers are unary encoded.

▶ **Lemma 10.** Multi-Way Number Partition *parameterized by the number of sets k is $W[1]$-hard. Moreover, unless ETH fails there is no algorithm that solves the problem in $f(k)N^{o(k/\log k)}$ time for some function f where N is the input size.*

▶ **Theorem 11.** *The* Exact Path Packing *problem parameterized by the number of paths on graphs with treewidth two is $W[1]$-hard. Moreover, unless ETH fails there is no algorithm that solves the problem in $f(k)\ n^{o(k/\log k)}$ time for some function f where k is the number of paths and n the number of vertices in the input graph.*

4 Path Packing Parametrized by Path Dependent Attributes

In the previous section we solved Path Packing on forests. Since Path Packing is NP-hard even for graphs with treewidth 2, we try to find some path dependent parameters to cope with its difficulty. At first, we will restrict the number of paths, then we will bound the length of each path and finally we consider the sum of the lengths of all paths.

Number of Paths

We denote the number of paths of an instance by k. We start with $k = 1$. Consider an instance where the length of the single path corresponds to the number of vertices in a complete graph G.

▶ **Observation 12.** *Since Hamiltonian Path is* NP*-hard, also path packing for $k = 1$ is* NP*-hard*

On the other side, for $k = 1$ the special case of Exact Path Packing becomes easy.

▶ **Observation 13.** Exact Path Packing *is solvable in polynomial time for $k = 1$ by deciding if the input graph is a path of length l_1.*

Unfortunately, for fixed $k \geq 2$ restricting the number of paths is not enough to gain a polynomial time algorithm. This holds for Exact Path Packing and therefore also for Path Packing.

▶ **Theorem 14.** *Let $k \geq 2$.* Exact Path Packing *with k paths is* NP*-complete on 4-regular graphs.*

Paths with bounded length

Observe that all hardness proofs that we have seen so far somehow involve paths of a certain length. Thus, we analyze the complexity of Path Packing based on the length of the paths we want to pack.

Length-i Exact Edge Packing

Input: A set of paths $P = \{p_1, \ldots, p_k\}$ of length $l_1 = \ldots = l_k = i$, a graph $G = (V, E)$.

Question: Is there an edge-disjoint embedding of P into G such that each edge $e \in E$ is covered exactly once?

Length-2 Exact Edge Packing is solvable in polynomial time by reformulating it as a matching problem on the line graph. We show that Length-3 Exact Edge Packing is already NP-hard via a reduction from the 3-dimensional matching problem, one of Karp's original 21 NP-complete problems. The 3-dimensional matching problem takes as input sets X, Y, Z of size n and $T \subseteq X \times Y \times Z$. The question is whether there exists a set $M \subseteq T$ of size n such that for all $(x_1, y_1, z_1), (x_2, y_2, z_2) \in M$, $x_1 \neq x_2$, $y_1 \neq y_2$, $z_1 \neq z_2$. The reduction is similar to the P_2-packing reduction in [20].

▶ **Theorem 15.** Length-3 Exact Edge Packing *is* NP*-hard.*

Bounded sum of path lengths

The two previous results show that Path Packing is NP-hard even if the number of paths or the maximal length of the paths is bounded by a constant. At last, we the problem is in FPT when parameterized by the number of paths *and* their length. We give a randomized FPT algorithm using color coding [8, 1] that can easily be derandomized using perfect hash families [8].

▶ **Theorem 16.** Path Packing *parameterized by the summed length of all paths is in* FPT.

For packing vertex-disjoint paths similar results are known: The P_2-packing problem takes a graph G and an integer k and asks if there is a set of k vertex-disjoint P_2 in G. This problem is NP-complete [16, 20]. Fernau and Raible given FPT algorithm parameterized by the number of paths [10].

5 Path Packing Parametrized by Graph Dependent Attributes

Earlier (Theorem 1), we showed that Exact Path Packing is NP-hard even on a single subdivided star. So even for trees where there is only one node of degree higher than two the problem becomes NP-hard. In this section we study further restrictions to forests and finally identify a polynomial time solvable case. We do so by considering restrictions to the following three parameters: number of vertices of degree at least three, the maximal degree, and the number of connected components. For an easier notation we define this combined parameter as the "bcd" of graph G. It is a bound on **b**ranching nodes, connected **c**omponents and maximum **d**egree.

▶ **Definition 17.** *Let G be a graph. Then bcd(G) is the minimal $k \in \mathbf{N}$ such that G has at most k nodes of degree larger than two, at most k connected components, and a maximal degree of at most k.*

The above mentioned reduction showing NP-hardness for a subdivided star constructs a graph with unbounded degree. What is the complexity if we limit the vertex degree to a constant, but in turn allow multiple components? Unfortunately even for a forest of paths, thus a maximum degree of two, the problem remains NP-hard. NP-hardness follows by an easy adaption of the proof of Theorem 1. The constructed subdivided star has an even number $2m$ of legs of length ℓ. Instead one could also use m disjoint paths of length 2ℓ. Thus we can follow NP-hardness of Exact Path Packing even for forests of paths.

▶ **Corollary 18.** Exact Path Packing *is* NP*-hard even on forests of paths.*

Thus, if we drop either the degree or the number of components as parameters, the problem becomes NP-complete, even if the remaining parameters are bounded by a constant. Thus the remaining question is: What is the complexity if we limit the vertex degree to a constant, limit the number of connected components to a constant, but in turn allow arbitrary many vertices of degree at least three? We show hardness in this scenario even for the more restricted problem of packing paths of ascending length, denoted by Path-Perfect Packing.

Path-Perfect Packing

Input:	A graph $G = (V, E)$ where $\lvert E(G)\rvert = 1 + 2 + \cdots + n$ for some $n \in \mathbf{N}$.
Question:	Does there exist an edge-disjoint embedding of paths $p_1, \ldots, p_n$ with lengths $\ell_1 = 1, \ldots, \ell_n = n$ into G such that each edge $e \in E$ is covered exactly once?

We show NP-hardness of this restricted problem on subdivisions of a caterpillar with vertex degree at most eight. We reduce from the following unary version of 3-partition. This version is slightly non-standard since we require that no numbers occur twice and relax the condition to put exactly three elements into each partition.

Unary 3-Partition
Input: A set of integers $A = \{a_1, \ldots, a_{3s}\} \subseteq \mathbf{N}$ in unary encoding.
Question: Is there a partition of A into s sets of equal sum?

Hulett et al. show that the above problem is NP-hard if we require each of the s partitions to contain exactly three elements [14]. We get NP-hardness without the extra condition by increasing each number in A by adding a big number (for example $\sum_{i=}^{3s} a_i$).

We sketch how to reduce from Path-Perfect Packing on caterpillars with vertex degree at most eight to Unary 3-Partition. Note that, each partition of a Unary 3-Partition instance must have size $m = \frac{1}{s}\sum_{i=1}^{3s} a_i$. Consider the paths $\{p_i \mid 1 \leq i \leq m,\ i \in A\}$ whose length occurs in A. We translate a partition of A into an exact packing of these paths. However, we have to account for the paths $\{p_i \mid 1 \leq i \leq m,\ i \notin A\}$ whose length occurs *not* as an integer in set A. For each of these we introduce a path where it fits in precisely, and by an exchange argument we may assume it is packed there. Now, roughly what remains to do is to construct a large caterpillar of low maximum degree where all these paths can be packed in.

▶ **Theorem 19.** Path-Perfect Packing *is* NP*-hard even for subdivided caterpillars with vertex degree at most eight.*

The maximum degree eight in the theorem above was chosen to simplify the proof. One can show NP-hardness even for smaller maximum vertex degree.

XP-algorithm for parameter bcd(G)

We saw that if two bcd-parameters are constant and one bcd-parameter is unbounded then Exact Path Packing is NP-complete. We further study the complexity when parameterized by all three parameters. We give an XP-algorithm for Path Packing parametrized by $\mathrm{bcd}(G)$. This means for every fixed k, there is a polynomial time algorithm for graphs with $\mathrm{bcd}(G) \leq k$.

▶ **Theorem 20.** *There is a* $k!^k(n+k^2)^{O(k^2)}$*-time algorithm for* Path Packing *with* $k = bcd(G)$.

Proof. We give an algorithm, that given a graph G and a list P of paths $p_1, \ldots, p_k$, decides if there is an edge-disjoint embedding of $p_1, \ldots, p_k$ into G. To do so, we guess a partition of G into eventually a set of vertex-disjoint paths $\mathcal{X}$. Then it suffices to find an embedding of P into such a set of vertex-disjoint paths $\mathcal{X}$. The remaining problem then is just a generalized bin-packing problem with $O(k^2)$ bins, but encoded in unary; thus solvable in time $n^{O(k^2)}$. Most technicality lies in guessing the vertex-disjoint paths $\mathcal{X}$. First we guess a partition into a bounded number of walks $\mathcal{W}$. Later we need to partition $\mathcal{W}$ further resulting in vertex-disjoint paths $\mathcal{X}$.

Let V_1 be the set of vertices of degree two. Let $V_2^\star$ consist of a vertex of every connected component that is a circle. Let V_2 be the set of vertices of degree two that are not in $V_2^\star$, and let $V_{\geq 3}$ be the vertices of degree at least three including $V_2^\star$. This seemingly odd definition allows us to work with walks starting and ending in $V_1 \cup V_{\geq 3}$ that cover every edge, in

particular those in a circle. Because there are at most k connected components, $|V_2^\star| \leq k$. Then since there are at most k vertices of degree at least three, we have $|V_{\geq 3}| \leq 2k$.

Assuming a yes-instance, there is an edge-disjoint embedding of paths P into graph G. At every vertex $v \in V_{\geq 3}$ every path of P contains at most two of the incident edges of v. Thus at every vertex $v \in V_{\geq 3}$ there is a maximal matching M_v of v's incident edges such that no path in P contains two unmatched edges.

We consider "direct" walks between "neighboring" vertices $V_1 \cup V_{\geq 3}$: Let $\mathcal{Q}$ be the set of walks between $u, v \in V_1 \cup V_{\geq 3}$ with inner vertices from V_2, and further where no vertex among V_2 is repeated (though possibly $u = v$). We join these walks $\mathcal{Q}$ to a set of walks $\mathcal{W}$ according to matchings M_v for $v \in V(G)$. Whenever two walks w_1, w_2 end at some edges uv respectively $u'v$, and uv is matched to $u'v$ by M_v, then join walks w_1 and w_2 at edges uv, vu'. This procedure terminates and yields a well defined set of walks $\mathcal{W}$.

Note that every edge is covered by a walk $\mathcal{Q}$ and thus also every edge is covered by a walk $\mathcal{W}$. We further claim that every path p of P is a subsequence of edges of some walk $w \in \mathcal{W}$. Assuming otherwise, there are walks w_1, w_2 ending at edges uv and $u'v$. Then v is not a leaf, and thus $v \in V_{\geq 3}$. Then matching M_v matches edges uv and $u'v$, and thus w_1, w_2 had to be joined to a single walk.

Thus for a yes-instance there is at least one set of matchings $M_v, v \in V_{\geq 3}$ which determines walks $\mathcal{W}$ such that P may be embedded into $\mathcal{W}$. An algorithm may try the possible partition of edges into such a set of walks $\mathcal{W}$ as follows. Guess for each vertex $v \in V_{\geq 3}$ a maximal matching M_v of its incident edges. There are at most k high degree vertices $V_{\geq 3} \setminus V_2^\star$, each with at most k incident edges. (Also we have a matching for $V_2^\star$, though since there are only two incident edges, there is only one possible matching.) Thus the algorithm tries at most $k!^k$ possibilities. Then combine the paths $\mathcal{Q}$ to walks $\mathcal{W}$ according to the matchings, which is possible in polynomial time.

We claim that $\mathcal{W}$ has at most k^2 walks. Every walk in $\mathcal{W}$ has two endpoints, and the endpoints are among $V_1 \cup V_{\geq 3}$. Clearly, at every leaf $v \in V_1$ at most one path ends. Further, there are at most k^2 leaves in the input graph of $\leq k$ vertices of degree ≥ 3 and maximal degree of k. If at a vertex $v \in V_{\geq 3}$ two walks w_1, w_2 end, there are edges uv of w_1 and $u'v$ of w_2 unmatched by M_v, in contradiction to a maximal matching M_v. Thus also at every vertex $v \in V_{\geq 3}$ at most one walk ends. Then there are at most $k^2 + 2k$ endpoints of walks, and thus there are at most $\lfloor (k^2 + 2k)/2 \rfloor \leq k^2$ walks in $\mathcal{W}$.

Consider a walk $w \in \mathcal{W}$ where a vertex v occurs more than once. Recall that the embedding of a path $p \in P$ of a yes-instance is injective, thus no vertex $v \in V(G)$ occurs twice in the same path. A naive approach would be to now solve the bin-packing problem of "weights" P and "bins" $\mathcal{W}$. Then, however, a solution to the bin-packing would may potentially translate to an embedding of a path where a vertex occurs twice. Therefore let us guess a partition into paths without multiple occurrence of vertices, as follows.

Between two occurrences of v on walk w there must be vertex u (possibly an occurrence of v itself) which is the endpoint of two different paths. Therefore there is a partition of the walks $\mathcal{W}$ into vertex-disjoint paths $\mathcal{X}$, where still paths P have an embedding into $\mathcal{X}$. We may describe this partition by "cuts" of $\mathcal{W}$ specified by a vertex v in the union of walks from $\mathcal{W}$. Note, that in the union of walks $\mathcal{W}$, each high degree vertex $v \in V_{\geq 3} \setminus V_2^\star$ occurs $\deg(v) \leq k$ times. Thus there are to up to $n + k^2$ potential cut vertices.

We claim that at most k^2 cuts C are necessary to cut the walks $\mathcal{W}$ into vertex-disjoint paths $\mathcal{X}$. Assume, that there is a cut vertex $v \in C$ which is on an inner vertex of a path between V_1 and $V_{\geq 3}$. Then joining its incident vertex-disjoint paths results in a vertex-disjoint path. Thus we may assume that the cuts C are at vertices from walks of $\mathcal{Q}$ between vertices

among $V_{\geq 3}$. Let $\mathcal{Q}_{\geq 3}$ be the set of paths $\mathcal{Q}$ with endpoints in $V_{\geq 3}$. Consider the multi-graph with loops on vertex set $V_{\geq 3}$ with an edge between $u, v \in V_{\geq 3}$ for every path $\mathcal{Q}_{\geq 3}$ with endpoints u and v. Since $|Q_{\geq 3}| \leq k$ and the degree of every vertex $v \in Q_{\geq 3}$ is at most k, this multi-graph has at most k^2 edges. Then also there are at most k^2 paths $\mathcal{Q}_{\geq 3}$. Consider a set of more than k^2 vertices $C \subseteq V$ that cut $\mathcal{Q}$ into vertex-disjoint paths $\mathcal{X}$. Then there is a path of $Q_{\geq 3}$ containing distinct cut vertices $u, v \in C$. Let $x \in \mathcal{X}$ be the path between u and v. Joining them with the incident path at, say u, results in a vertex-disjoint path. Thus cutting $\mathcal{W}$ at vertices $C \setminus \{u\}$ still results in a set of vertex-disjoint paths. Therefore at most k^2 cuts of walks $\mathcal{W}$ are necessary to yield vertex-disjoint paths $\mathcal{X}$.

Let us utilize this observation in the design of our algorithm. Guess up to k^2 cuts C from the $n + k^2$ potential cuts. Cut the previously guessed walks $\mathcal{W}$ into subpaths according to cuts C. We may force exactly k^2 cut vertices by allowing C to be a multi-set containing also leaves, whose cut has no effect. This way, we try another $(n+k^2)^{k^2}$ possibilities of cut vertices C. Then cut the previously guessed $\leq k^2$ walks $\mathcal{X}$ at the k^2 cut positions. If the resulting set of walks is not vertex-disjoint, discard this guess. Otherwise we obtain $\leq 2k^2$ vertex disjoint paths $\mathcal{X}$, since every cut increases the number of paths by one. This resembles a bin-packing problem in unary encoding with k^2 bins of different sizes and total capacity n. We may apply standard dynamic programming technique to test in $n^{O(k^2)}$ time whether the sizes of the paths P fit into the bins in the sizes of $\mathcal{X}$. If the paths P fit in some guessed paths $\mathcal{X}$, then corresponding partition of the edges in G yields paths P. Thus there is an edge-disjoint embedding of P into G. For the other direction, if the edges of G can be partitioned into paths P, then as argued before there is a set $\mathcal{X}$ according to this partition and the there is a solution to the dynamic problem. The runtime is $k!^k(n + k^2)^{O(k^2)} \cdot \text{poly}(n) = k!^k n^{O(k^2)} \cdot \text{poly}(n)$ where poly is a polynomial. ◀

Can we achieve a better runtime than $k!^k n^{k^2+O(1)}$, in particular decrease the dependence on k in the exponent of n? Not significantly unless ETH fails, as the following reduction from Multi-Way Number Partition shows.

▶ **Theorem 21.** *There is no algorithm that decides* Path Packing *in time* $n^{o(k^2/\log k)}$ *with* $k = bcd(G)$ *unless ETH fails.*

6 Conclusion

We showed that edge-disjoint packing of paths into a graph is a very hard problem. Even if the input graph is a subdivided star or a linear forest the problem is hard. If we parameterize the problem by the number of paths, the problem remains hard even for input graphs with treewidth two. However, it becomes fixed parameter tractable on forests. A natural open problem is to not embed paths, but more general graphs such as trees or cycles.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995. doi:10.1145/210332.210337.

2 S. Rao Arikati and C. Pandu Rangan. Linear Algorithm for Optimal Path Cover Problem on Interval Graphs. *Inf. Process. Lett.*, 35(3):149–153, July 1990. doi:10.1016/0020-0190(90)90064-5.

3 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: Fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74, 2007. doi:10.1145/1250790.1250801.

4 Maurizio A. Bonuccelli and Daniel P. Bovet. Minimum Node Disjoint Path Covering for Circular-Arc Graphs. *Inf. Process. Lett.*, 8(4):159–161, 1979. doi:10.1016/0020-0190(79)90011-5.

5 Weiting Cao and Peter Hamburger. Solution of fink & straight conjecture on path-perfect complete bipartite graphs. *Journal of Graph Theory*, 55(2):91–111, 2007.

6 Gerard J. Chang and David Kuo. The $L(2,1)$-Labeling Problem on Graphs. *SIAM Journal on Discrete Mathematics*, 9(2):309–316, 1996. URL: https://epubs.siam.org/doi/ref/10.1137/S0895480193245339.

7 Gerard Cornuéjols, David Hartvigsen, and W Pulleyblank. Packing subgraphs in a graph. *Operations Research Letters*, 1(4):139–143, 1982.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

9 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

10 Henning Fernau and Daniel Raible. A parameterized perspective on packing paths of length two. In *International Conference on Combinatorial Optimization and Applications*, pages 54–63. Springer, 2008.

11 John Frederick Fink and H. Joseph Straight. A note on path-perfect graphs. *Discrete Mathematics*, 33(1):95–98, 1981.

12 S. E. Goodman, Stephen T. Hedetniemi, and Peter J. Slater. Advances on the Hamiltonian Completion Problem. *J. ACM*, 22(3):352–360, 1975. doi:10.1145/321892.321897.

13 Peter Hamburger and Weiting Cao. Edge Disjoint Paths of Increasing Order in Complete Bipartite Graphs. *Electronic Notes in Discrete Mathematics*, 22:61–67, 2005.

14 Heather Hulett, Todd G. Will, and Gerhard J. Woeginger. Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Operations Research Letters*, 36(5):594–596, 2008. doi:10.1016/j.orl.2008.05.004.

15 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013.

16 David G. Kirkpatrick and Pavol Hell. On the Completeness of a Generalized Matching Problem. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 240–245, 1978. doi:10.1145/800133.804353.

17 Hoàng-Oanh Le, Van Bang Le, and Haiko Müller. Splitting a graph into disjoint induced paths or cycles. *Discrete Applied Mathematics*, 131(1):199–212, 2003. doi:10.1016/S0166-218X(02)00425-0.

18 Jayadev Misra and Robert Endre Tarjan. Optimal Chain Partitions of Trees. *Inf. Process. Lett.*, 4(1):24–26, 1975. doi:10.1016/0020-0190(75)90057-5.

19 Jerome Monnot and Sophie Toulouse. The path partition problem and related problems in bipartite graphs. *Operations Research Letters*, 35(5):677–684, 2007. doi:10.1016/j.orl.2006.12.004.

20 Sarah Pantel. Graph packing problems. Master's thesis, Simon Fraser University, Canada, 1999.

21 R. Srikant, Ravi Sundaram, Karan Sher Singh, and C. Pandu Rangan. Optimal Path Cover Problem on Block Graphs and Bipartite Permutation Graphs. *Theor. Comput. Sci.*, 115(2):351–357, 1993. doi:10.1016/0304-3975(93)90123-B.

22 George Steiner. On the k-path partition of graphs. *Theoretical Computer Science*, 290(3):2147–2155, 2003. doi:10.1016/S0304-3975(02)00577-7.

23 H. Joseph Straight. *Partitions of the vertex set or edge set of a graph*. dissertation, Western Michigan University, 1977.

24 H. Joseph Straight. Packing trees of different size into the complete graph. *Annals of the New York Academy of Sciences*, 328(1):190–192, 1979.

25 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In *Proc. of the 17th*

Annual European Symposium on Algorithms (ESA), number 5757 in Lecture Notes in Computer Science, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

26 Jing-Ho Yan and Gerard J. Chang. The path-partition problem in block graphs. *Information Processing Letters*, 52(6):317–322, 1994. doi:10.1016/0020-0190(94)00158-8.

27 Jing-Ho Yan, Gerard J. Chang, Sandra Mitchell Hedetniemi, and Stephen T. Hedetniemi. k-Path Partitions in Trees. *Discrete Applied Mathematics*, 78(1-3):227–233, 1997. doi:10.1016/S0166-218X(97)00012-7.

28 H. P. Yap. Packing of graphs-a survey. In *Annals of Discrete Mathematics*, volume 38, pages 395–404. Elsevier, 1988.

29 Shmuel Zaks and Chung Laung Liu. Decomposition of graphs into trees. Technical Report 860, Department of Computer Science, University of Illinois at Urbana-Champaign, 1977.