On Kernels for d-Path Vertex Cover

Radovan Červený
Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic

Pratibha Choudhary
Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic

Ondřej Suchý
Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic

Abstract

In this paper we study the kernelization of the d-Path Vertex Cover (d-PVC) problem. Given a graph G, the problem requires finding whether there exists a set of at most k vertices whose removal from G results in a graph that does not contain a path (not necessarily induced) with d vertices. It is known that d-PVC is NP-complete for $d \geq 2$. Since the problem generalizes to d-Hitting Set, it is known to admit a kernel with $\mathcal{O}(dk^d)$ edges. We improve on this by giving better kernels. Specifically, we give kernels with $\mathcal{O}(k^2)$ vertices and edges for the cases when $d = 4$ and $d = 5$. Further, we give a kernel with $\mathcal{O}(k^4 d^{2d+9})$ vertices and edges for general d.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases Parameterized complexity, Kernelization, d-Hitting Set, d-Path Vertex Cover, Expansion Lemma

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.29

Related Version *Full Version*: `https://arxiv.org/abs/2107.12245`

Funding *Radovan Červený*: The author acknowledges the support of the Grant Agency of the Czech Technical University in Prague, grant No. SGS20/208/OHK3/3T/18.
Pratibha Choudhary: The author acknowledges the support of the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".
Ondřej Suchý: The author acknowledges the support of the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

1 Introduction

Vertex deletion problems have been studied extensively in graph theory. These problems require finding a subset of vertices whose deletion results in a graph that belongs to some desired class of graphs. One such problem is path covering. Given a graph $G = (V, E)$, the d-Path Vertex Cover problem (d-PVC) asks to compute a subset $S \subseteq V$ of vertices such that the graph resulting from removal of S does not contain a path on d vertices. Here the path need not necessarily be induced. The problem was first introduced by Brešar et al. [1]. It is known to be NP-complete for any $d \geq 2$ due to the meta-theorem of Lewis and Yannakakis [21]. The 2-PVC problem is the same as the well known Vertex Cover problem. The 3-PVC problem is also known as Maximum Dissociation Set or Bounded Degree-One Deletion. The d-PVC problem is motivated by the field of designing secure wireless communication protocols [22] or in route planning and speeding up of shortest path queries [18].

With respect to exact algorithms, several efficient (better than brute force enumeration) algorithms are known for 2-PVC and 3-PVC. In particular, 2-PVC (Vertex Cover) can be solved in $\mathcal{O}(1.1996^n)$ time and polynomial space due to Xiao and Nagamochi [33], while 3-PVC can be solved in $\mathcal{O}(1.4613^n)$ time and polynomial space due to Chang et al. [6] or in $\mathcal{O}(1.3659^n)$ time and exponential space due to Xiao and Kou [31].

From the approximation point of view, it is known due to Brešar et al. [1] that d-PVC, for $d > 2$, cannot be r-approximated within a factor of $r = 1.3606$ in polynomial time, unless P=NP. A greedy d-approximation algorithm for d-PVC can be employed by repeatedly finding a d-path and putting its vertices into the solution. Due to Fomin et al. [17], we can find an arbitrary d-path in $\mathcal{O}(2.619^d dn \log n)$ time, and therefore the approximation algorithm runs in $\mathcal{O}(n^2 \log n)$ time in the size of the input. While the algorithms of Zehavi [34] and Tsur [25], with running times $\mathcal{O}^*(2.597^d)$ and $\mathcal{O}^*(2.554^d)$,[1] respectively, can be faster for large d, their running time factor polynomial in input size is much worse than $\mathcal{O}(n \log n)$. Lee [20] gave a $\mathcal{O}(\log d)$-approximation algorithm which runs in $\mathcal{O}^*(2^{\mathcal{O}(d^3 \log d)})$ time. For 3-PVC a 2-approximation algorithm was given by Tu and Zhou [30] and for 4-PVC a 3-approximation algorithm is known due to Camby et al. [2].

When parameterized by the size of the solution k, d-PVC is directly solvable by a trivial FPT algorithm for d-Hitting Set, that runs in $\mathcal{O}^*(d^k)$ time. However, since d-PVC is a restricted case of d-Hitting Set, it is known due to Fomin et al. [15] that for $d \geq 4$ d-PVC can be solved in $\mathcal{O}^*((d - 0.9245)^k)$ time and for $d \geq 6$ algorithms with even better running times are known due to Fernau [14]. Namely the running times are $\mathcal{O}^*((d - 1 + c_d)^k)$, where c_d is a small positive constant which monotonically approaches 0 as d goes to ∞. There has been considerable study for the case when d is a small constant. For the 2-PVC (Vertex Cover) problem, current best known algorithm due to Chen, Kanj, and Xia [7] runs in time $\mathcal{O}^*(1.2738^k)$. For 3-PVC, the current best known algorithm due to Tsur [27] runs in $\mathcal{O}^*(1.713^k)$ time. For the 4-PVC problem, Tsur [28] gave the current best algorithm that runs in $\mathcal{O}^*(2.619^k)$ time. In previous work [3, 4], a subset of authors developed an $\mathcal{O}^*(4^k)$ algorithm for 5-PVC. For $d = 5$, 6, and 7 Tsur [26] claimed algorithms for d-PVC with running times $\mathcal{O}^*(3.945^k)$, $\mathcal{O}^*(4.947^k)$, and $\mathcal{O}^*(5.951^k)$, respectively. A subset of authors used a computer to generate even faster algorithms for $3 \leq d \leq 8$ [5].

In this paper, we are interested in kernels for the d-PVC problem. Since an instance of d-PVC can be formulated as an instance of d-Hitting Set, by using the results of Fafianie and Kratsch [13] we immediately get a kernel for d-PVC with at most $d(k+1)^d$ vertices and at most $(d-1)(k+1)^d$ edges by keeping only the vertices and edges that are contained in the corresponding sets of the reduced d-Hitting Set instance.

Regarding the lower bounds for kernels of d-PVC, Dell and Melkebeek [10] have shown that for Vertex Cover it is not possible to achieve a kernel with $\mathcal{O}(k^{2-\varepsilon})$ edges unless coNP is in NP/poly (which would imply a collapse of the polynomial hierarchy). This result extends to d-PVC for any $d \geq 2$. Therefore, kernels with $\mathcal{O}(k^2)$ edges for d-PVC are the best we can hope for.

The current best kernels known are a kernel for Vertex Cover with $2k - c \log k$ vertices for any fixed constant c due to Lampis [19] and a kernel with $5k$ vertices for 3-PVC due to Xiao and Kou [32]. No specific kernels are known for d-PVC with $d \geq 4$, except for those inherited from d-Hitting Set.

Dell and Marx [9] recently studied kernels for the related d-Path Packing problem, which also inspired our work.

[1] The $\mathcal{O}^*()$ notation suppresses all factors polynomial in the input size.

Our contribution

We give kernels with $\mathcal{O}(k^2)$ edges for 4-PVC and 5-PVC (asymptotically optimal, unless coNP $\subseteq$ NP/poly). Furthermore, for the general case, we give a kernel for d-PVC for any $d \geq 6$ with $\mathcal{O}(k^4 d^{2d+9})$ edges.

2 Preliminaries

We use the notations related to parameterized complexity as described by Cygan et al. [8]. We consider simple and undirected graphs unless otherwise stated. For a graph G, we use $V(G)$ to denote the vertex set of G and $E(G)$ to denote the edge set of G. By $G[X]$ we denote the subgraph of G induced by vertices of $X \subseteq V(G)$. By $N(v)$ we denote the set of neighbors of $v \in V(G)$ in G. Analogically, $N(X) = \bigcup_{x \in X} N(x) \setminus X$ denotes the set of neighbors of vertices in $X \subseteq V(G)$. The degree of vertex v is denoted by $\deg(v) = |N(v)|$. For simplicity, we write $G \setminus v$ for $v \in V(G)$ and $G \setminus X$ for $X \subseteq V(G)$ as shorthands for $G[V(G) \setminus \{v\}]$ and $G[V(G) \setminus X]$, respectively.

A *d-path* (also denoted by P_d), denoted as an ordered d-tuple $(p_1, p_2, \ldots, p_d)$, is a path on d vertices $\{p_1, p_2, \ldots, p_d\}$. A *$d$-path free* graph is a graph that does not contain a d-path as a subgraph (the d-path needs not to be induced). The *length* of a path P is the number of edges in P, in particular, the length of a d-path P_d is $d - 1$.

The d-Path Vertex Cover problem is formally defined as follows:

d-Path Vertex Cover, d-PVC	
Input:	A graph $G = (V, E)$, a non-negative integer k.
Output:	A set $S \subseteq V$, such that $\lvert S \rvert \leq k$ and $G \setminus S$ is a P_d-free graph.

A d-path packing $\mathcal{P}$ of size l in a graph G is a collection of l vertex disjoint d-paths in the graph G. We use $V(\mathcal{P})$ to denote the union of the vertex sets of the d-paths in the packing $\mathcal{P}$. For rest of the graph theory notations we refer to Diestel [11].

For a positive integer i, we will use $[i]$ to denote the set $\{1, 2, \ldots, i\}$.

▶ **Proposition 1** (⋆). *For a given graph G and an integer k, there is an algorithm which either correctly answers whether G has a d-path vertex cover of size at most k, or finds an inclusion-wise maximal d-path packing $\mathcal{P}$ of size at most k in $\mathcal{O}(2.619^d dkn \log n)$ time.*

3 General Reduction Rules

Let us start with reduction rules that apply to d-PVC for most values of d. Assume that we are working with an instance $(G = (V, E), k)$ of d-PVC for some $d \geq 4$. We start with a reduction rule whose correctness is immediate.

▶ **Reduction Rule 1.** *If there is a connected component C in G which does not contain a P_d, then remove C.*

The next rule allows us to get rid of multiple degree-one vertices adjacent to a single vertex.

▶ **Reduction Rule 2** (⋆). *Let there be three distinct vertices $v, x, y \in V$ such that $N(x) = N(y) = \{v\}$. We reduce the instance by deleting the vertex x.*

4 High Degree Reduction Rule for 4-PVC and 5-PVC

In this section, we are going to introduce the reduction rules which are applicable to both 4-PVC and 5-PVC instances. We assume that we are working with a d-PVC instance $(G = (V, E), k)$ for $d \in \{4, 5\}$ which is reduced by exhaustively employing Reduction Rule 2.

Our aim is to show that the degree of each vertex can be reduced to linear in the parameter. First assume that there is a large matching in the neighborhood of some vertex v. We call a matching $\mathcal{M}$ in G *adjacent* to vertex v, if it is a matching in $G \setminus v$ and for each edge $\{a_i, b_i\} \in \mathcal{M}$ at least one of its vertices, say a_i, is adjacent to v in G.

▶ **Reduction Rule 3** ($\star$). *If v is a vertex and $\mathcal{M}$ a matching adjacent to v of size $|\mathcal{M}| \geq k+2$, then delete v and decrease k by 1.*

To exhaustively apply Reduction Rule 3, we need to find for each $v \in V$ a largest matching adjacent to v. This can be done as follows. Let $A = N(v)$ and $B = N(A) \setminus \{v\}$. Let G_v be the graph obtained from $G[A \cup B]$ by removing edges with both endpoints in B. It is easy to observe, that each matching adjacent to v is also a matching in G_v and vice-versa. Hence, it suffices to find a largest matching in G_v, which can done in polynomial time [12].

Therefore, we further assume that the instance is reduced with respect to Reduction Rule 3. We fix a vertex v and find a largest matching $\mathcal{M}$ adjacent to it by the above algorithm. Let M be the set of vertices covered by matching $\mathcal{M}$ and $m = |\mathcal{M}|$. Since the instance is reduced, we know that $m \leq k + 1$. Let $X = N(v) \setminus M$. We refer the reader to the Figure 1 for overview of our setting.

▶ **Observation 2** ($\star$). *For each $x \in X$ we have $N(x) \setminus \{v\} \subseteq M$.*

▶ **Observation 3** ($\star$). *No two distinct vertices $x, y \in X$ are connected to the opposite endpoints of a single edge $\{a_i, b_i\}$ in $\mathcal{M}$.*

▶ **Observation 4** ($\star$). *If there is a vertex $x \in X$ such that for some edge $\{a_i, b_i\}$ in the matching $\mathcal{M}$ we have that $\{a_i, b_i\} \subseteq N(x)$, then $N(\{a_i, b_i\}) \cap X = \{x\}$.*

We now partition the set X into three sets. Let X_2 be the set of vertices such that for each $x \in X_2$ we have some edge $\{a_i, b_i\}$ in the matching $\mathcal{M}$ such that $\{a_i, b_i\} \subseteq N(x)$. Let X_0 be the vertices such that for each $x \in X_0$ we have that $N(x) = \{v\}$. Note, that X_0 contains at most one vertex due to Reduction Rule 2 being exhaustively applied. Lastly, let $X_1 = X \setminus (X_2 \cup X_0)$ be the rest of the vertices in X. See Figure 1 for an illustration of the sets X_2, X_0, and X_1.

▶ **Observation 5** ($\star$). *If the vertex v has degree at least $(d + 2)(k + 1) + 1$, then $|X_1| \geq (d - 1)(k + 1)$.*

Now we focus on the edges between X_1 and M. By Observation 3, for each edge $\{a_i, b_i\}$ in $\mathcal{M}$ we have that the vertices in X_1 may be adjacent to at most one vertex of such edge, i.e. $|\{a_i, b_i\} \cap N(X_1)| \leq 1$. Letting $M_1 = M \cap N(X_1)$ we have $|M_1| \leq k + 1$.

We are now ready to employ the Expansion Lemma. We use the version of Fomin et al. [16], which is a generalization of the original results by Prieto [23, Corollary 8.1] and Thomassé [24, Theorem 2.3].

▶ **Definition 6.** *Let G be a bipartite graph with vertex bipartition (A, B). A set of edges $Q \subseteq E(G)$ is called a q-expansion, $q \geq 1$, of A into B if every vertex of A is incident with exactly q edges of Q, and Q saturates exactly $q|A|$ vertices in B.*

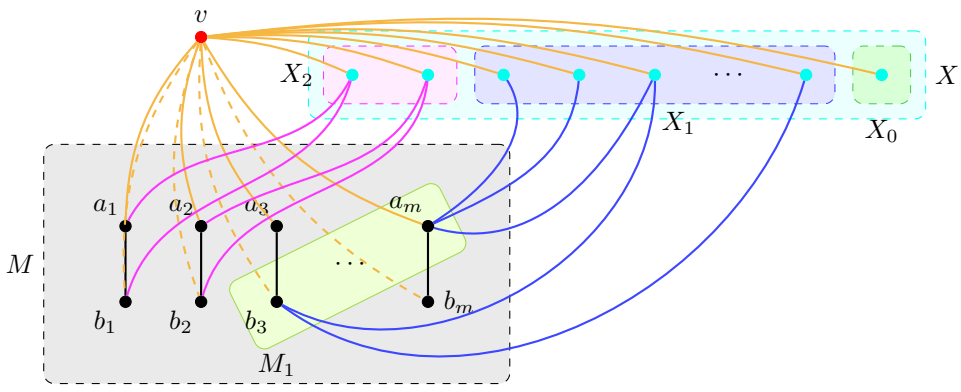

Figure 1 An overview of the definitions of sets X_0, X_1, X_2, and M_1.

▶ **Lemma 7** (Expansion Lemma; Fomin et al. [16]). *Let q be a positive integer, and G be a bipartite graph with bipartition (A, B) such that $|B| \geq q|A|$, and there are no isolated vertices in B. Then, there exists nonempty $A' \subseteq A$ and $B' \subseteq B$ such that A' has a q-expansion into B' and $N(B') \subseteq A'$. Moreover, the sets A', B' and the q-expansion can be found in polynomial time.*

▶ **Observation 8** (⋆). *There exist non-empty subsets $M' \subseteq M_1$ and $X' \subseteq X_1$ such that there is a $(d-1)$-expansion Q' from M' into X' and $N(X') \subseteq M' \cup \{v\}$.*

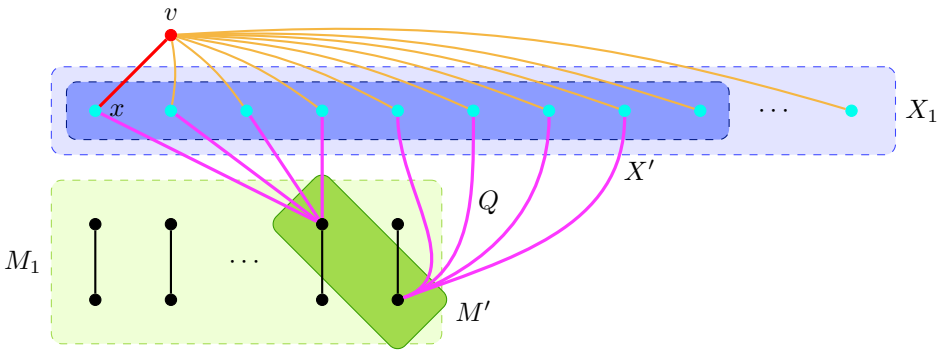

Figure 2 A graphical interpretation of applying the Expansion Lemma to our setting.

We refer the reader to Figure 2 for a graphical interpretation of the situation guaranteed by Observation 8. Now, let us focus on the sets M' and X' and the way they are connected with vertex v. We are going to show that some edge between v and X' is now redundant.

▶ **Reduction Rule 4.** *Let v be a vertex of degree at least $(d+2)(k+1)+1$. Let $\mathcal{M}$ be a largest matching adjacent to v and M be the set of vertices covered by $\mathcal{M}$. Let $X_1 \subseteq N(v) \setminus M$ be the set of vertices x with $N(x) \cap M \neq \emptyset$ and $|N(x) \cap \{a_i, b_i\}| \leq 1$ for each $\{a_i, b_i\} \in \mathcal{M}$. Let $M_1 = M \cap N(X_1)$. Let the non-empty subsets $M' \subseteq M_1$ and $X' \subseteq X_1$ be the sets with the $(d-1)$-expansion Q from M' into X' and such that $N(X') \subseteq M' \cup \{v\}$. Let $x \in X'$. Reduce the instance by deleting the edge $\{x, v\}$.*

Proof of Correctness. Let $(G = (V, E), k)$ be the original instance and $(G' = (V, E'), k)$ the reduced one. For each vertex $m \in M'$ let Q_m be the set of vertices of X' incident to m in the $(d-1)$-expansion Q. Since G' is a subgraph of G, if S is a solution for G, then S is also a solution for G'. Hence we will concetrate on the other direction.

Suppose that S' is a solution for the reduced instance. If it is also a solution for the original one, then we are done. Suppose it is not, i.e., there is a P_d in $G \setminus S'$. This P_d contains the edge $\{x, v\}$, otherwise it would be also present in $G' \setminus S'$. Therefore $v \notin S'$ and $x \notin S'$.

There are three ways how solution S' can interact with M' and X' that we need to address.

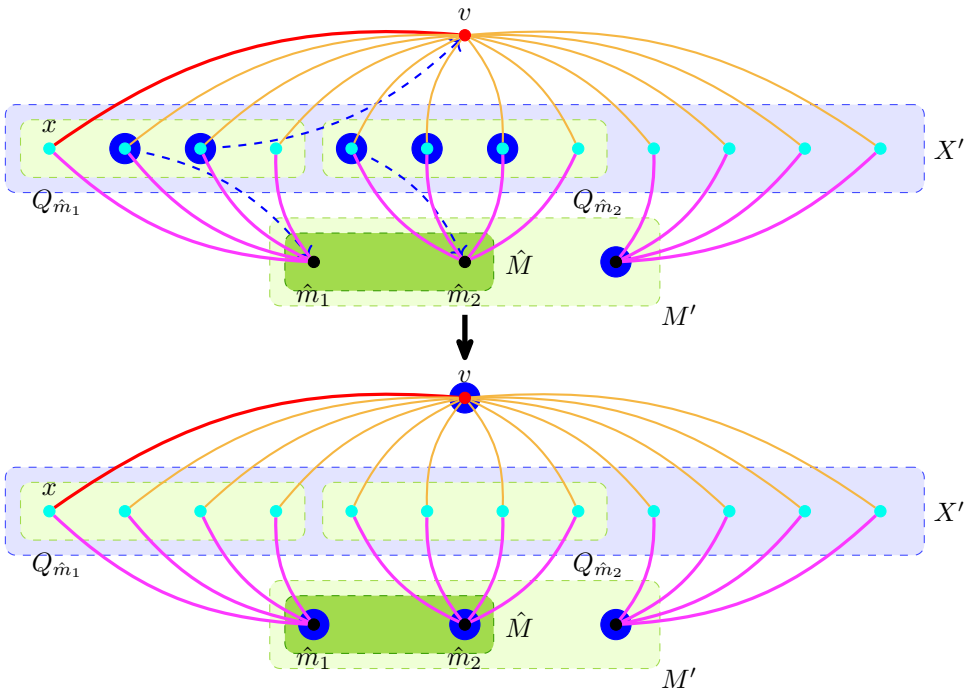

Figure 3 Illustration of the first part of the proof of correctness of Reduction Rule 4.

Firstly, suppose that $M' \not\subseteq S'$ and let $\widehat{M} = M' \setminus S'$. See Figure 3 for an illustration. We have that for each vertex $\widehat{m} \in \widehat{M}$ it must be that $|Q_{\widehat{m}} \cap S'| \geq 2$. Indeed, if this is not the case, there would be a P_d in $G' \setminus S'$ which uses the $(d-2)$ vertices of $Q_{\widehat{m}}$ not in S' and the vertices $\widehat{m}$ and v. Consider a set $S = (S' \cup \widehat{M} \cup \{v\}) \setminus \bigcup_{\widehat{m} \in \widehat{M}} Q_{\widehat{m}}$. Observe, that any P_d which uses some vertex from X' must contain at least one of the vertices in M' or v, because $N(X') \subseteq M' \cup \{v\}$. Therefore the set S is a solution for the reduced graph G' as it contains both M' and v. The set S is also a solution for G as it contains v and therefore covers any P_d which might use the deleted edge $\{x, v\}$. Finally, $|S| \leq |S'|$, because for each $\widehat{m}$ that we add into S, we remove at least two vertices of $Q_{\widehat{m}}$ from S and therefore we have that $|\widehat{M} \cup \{v\}| \leq 2|\widehat{M}| \leq |\bigcup_{\widehat{m} \in \widehat{M}} Q_{\widehat{m}}|$.

Secondly, assume that $M' \subseteq S'$ and there is some $x' \in X'$ such that $x' \in S'$. We construct the set $S = (S' \setminus X') \cup \{v\}$. Again, observe that S' is a solution for G' because any P_d which uses some vertex from X' must contain at least one of the vertices in M' or v and both are fully contained in S. We also have that S is a solution for G, again, as it contains v and therefore covers any P_d which might use the deleted edge $\{x, v\}$. Finally, $|S| \leq |S'|$, because we have the assumption that there is some $x' \in X'$ and $x' \in S'$.

Lastly, assume that $M' \subseteq S'$ and $X' \cap S' = \emptyset$. In this case, the P_d that we found in $G \setminus S'$ must be of the form $P = (x, v, u_3, \ldots, u_d)$ and $u_3, \ldots, u_d \notin (M' \cup X')$. The existence of such P_d also gives us that $v, u_3, \ldots, u_d \notin S'$. Let x' be an arbitrary vertex of $X' \setminus \{x\}$ (note that $|X'| \geq (d-1)|M'| \geq 3$). Then the P_d of the form $P' = (x', v, u_3, \ldots, u_d)$ can be found in both G and G', which contradicts the fact that S' is a solution in G'.

To sum up, we have shown that when we delete the edge $\{x, v\}$ from G, then for any solution S' for G' which is not also a solution for G, we can always find a new solution S, $|S| \leq |S'|$ which is a solution for both G' and G. ◀

As the application of the rule only requires finding a largest matching adjacent to v, classifying the vertices of $N(v)$, and finding a $(d-1)$-expansion and these tasks can be done in polynomial time, the rule can be applied in polynomial time.

5 4-PVC Kernel with Quadratic Number of Edges

Let $(G = (V, E), k)$ be an instance reduced by exhaustively employing Reduction Rules 1–4. Then the maximum degree in G is at most $(d+2)(k+1) = 6k+6$. Furthermore, assume that the algorithm of Proposition 1 actually returned an inclusion-wise maximal packing $\mathcal{P}$ in G with at most k 4-paths instead of answering immediately. Let $P = V(\mathcal{P})$, and let $A = V \setminus P$. There are at most $4k$ vertices in P. Each connected component in $G[A]$ is a 4-path free graph, otherwise we would be able to increase the size of the packing $\mathcal{P}$. Since the instance is reduced with respect to Reduction Rule 1, each connected component in $G[A]$ is connected to P by at least one edge.

To show that an instance reduced with respect to all the above rules has a quadratic number of edges, it suffices to count separately the number of edges incident on P and the number of edges in $G[A]$. Since the maximum degree in G is at most $6k+6$ and there are at most $4k$ vertices in P, there are at most $4k \cdot (6k+6) = 24k^2 + 24k$ edges incident on P.

To count the edges in $G[A]$, we first observe that a connected 4-path free graph is either a triangle, or a star (possibly degenerate, i.e., with at most 3 vertices). Here a *q-star* is a graph with vertices $\{c, l_1, \ldots, l_q\}$, $q \geq 0$ and edges $\{\{c, l_i\} \mid i \in \{1, \ldots, q\}\}$. Vertex c is called *a center*, vertices $\{l_1, \ldots, l_q\}$ are called *leaves*. The term *star* will be used for a q-star with an arbitrary number of leaves. Note that, a graph with a single vertex is a 0-star, a graph with two vertices and a single edge is a 1-star, and a 3-path is a 2-star. A *triangle* is a cycle on three vertices.

Secondly, as the instance is reduced with respect to Reduction Rule 1, each connected component in $G[A]$ is connected to P by at least one edge. Therefore, there are at most $24k^2 + 24k$ connected components in $G[A]$, as there are only that many edges going from P to A. Next, we provide an observation about stars in $G[A]$.

▶ **Observation 9** (⋆). *For each q-star $(c, l_1, l_2, \ldots, l_q)$ in $G[A]$, there are at most two vertices in the q-star which are not connected to P by any edge in G, one possibly being the center c and the other possibly being a leaf l_i of the star.*

▶ **Observation 10** (⋆). *There are at most $72k^2 + 72k$ edges in $G[A]$.*

We conclude this section with the final statement about our kernel.

▶ **Theorem 11** (⋆). *4-Path Vertex Cover admits a kernel with $96k^2 + 96k$ edges, where k is the size of the solution.*

6 5-PVC Kernel with Quadratic Number of Edges

The idea is completely analogous to the previous section. We employ the following characterization.

A *star with a triangle* is formed by connecting two leaves of a star with an edge. A *bi-star* is formed by connecting the centers of two stars with an edge.

▶ **Lemma 12** (Červený and Suchý [3, Lemma 4]). *A connected 5-path free graph is either a graph on at most 4 vertices, a star with a triangle, or a bi-star.*

We conclude this section with the final statement about our kernel.

▶ **Theorem 13** (⋆). *5-Path Vertex Cover admits a kernel with $245k^2 + 245k$ edges, where k is the size of the solution.*

7 d-PVC Kernel with $\mathcal{O}(k^4 d^{2d+9})$ Edges

In this section, we give a kernelization algorithm for d-PVC with $d \geq 6$.

An intuition behind the approach. The kernelization algorithm marks some vertices and edges, which it wants to keep, and throws away the rest. Essentially, the kernelization creates a subgraph $\widehat{G}$ of the input graph G. For correctness of the algorithm, we want to show that if there is a d-path P in G which misses some set of vertices S (a prospective solution), then we will also find some d-path P' in $\widehat{G}$, which also misses the set S.

We begin by finding a maximal packing $\mathcal{M}$ in G and we keep in $\widehat{G}$ all vertices M of the packing and all edges between them. Now, on one hand, if the path P would be completely contained in M, then trivially the path appears also in $\widehat{G}$. On the other hand, the path P cannot be completely outside of M. Thus, the path P crosses between M and outside of M at least once. This corresponds to vertices of M being connected by a path of prescribed length outside of M. We later formalize this as a "request".

To get more structure, we leverage the behavior of DFS trees of the connected components outside of M. With the DFS trees we identify vertices, which are "crucial" for the requests, and we further split the requests into "sub-requests" according to the "crucial" vertices.

The algorithm is inspired by Dell and Marx [9]. However, while the considered problems have similarities, many ideas are not translatable. In particular, they could afford to consider all "sub-requests" and keep $\Omega(k)$ vertices for each without affecting their bound (cf. [9, p. 23]). We had to be more careful in which "sub-requests" we consider and we need to employ Lemma 15 (below) to only keep $d^{\mathcal{O}(d)}$ vertices and edges for each such "sub-request" to achieve our precise bound. Also, to achieve the edge bound, we need to keep track of the purpose for which the individual vertices were marked, which makes it hard to split the algorithm into small self-contained steps.

Formal definitions. More formally, assume that we are given an instance $(G = (V, E), k)$ of d-PVC. We start by running the algorithm of Proposition 1 on the instance. If it answers directly, then we are done. Otherwise it returns a maximal packing $\mathcal{M}$ in G. Let M be the vertices of the packing $\mathcal{M}$. Recall that $|M| \leq dk$.

Let $G' = G \setminus M$, i.e., G' is the graph outside the packing $\mathcal{M}$. Label the connected components of G' as $G'_1, G'_2, \ldots, G'_t$. For each component G'_i pick an arbitrary vertex $r_i \in G'_i$ and compute a *depth-first search* tree T_i of G'_i rooted at r_i. Note that $V(T_i) = V(G'_i)$. Let $\mathcal{F}$ denote the forest consisting of all the trees $T_i, i \in [t]$. Note that $V(\mathcal{F}) = V(G')$.

For a rooted forest F and its vertex $v \in V(F)$, $sub(v)$ denotes the set of vertices of a maximal subtree of F rooted at v and $anc(v)$ the set of ancestors of v in F, i.e., $anc(v) = \{u \mid u \in V(F), v \in sub(u)\}$. Note that $v \in anc(v)$.

We provide the following observations regarding the DFS trees T_i and the forest $\mathcal{F}$.

▶ **Observation 14** ($\star$).

(a) *Each $y \in V(\mathcal{F})$ has at most $d-1$ ancestors.*

(b) *For two vertices $u, v \in V(\mathcal{F})$ which are not in ancestor-descendant relation, we have $sub(u) \cap sub(v) = \emptyset$ and $\{u, v\} \notin E(G)$.*

(c) *If C is a connected subgraph of G', then there is a vertex $w \in V(C)$ such that $V(C) \subseteq sub(w)$.*

A triple (f, l, X) is *an X-request*, if $l \in \{1, \ldots, d-1\}$, $X \subseteq V(G)$, and either $f = \{u, v\} \subseteq X$, $u \neq v$, or $f = \{x\} \subseteq X$. For an X-request (f, l, X) and $H \subseteq V(G)$, we use $\mathcal{P}^H_{f,l}$ to denote the set of paths P of length l in $G[H \cup f]$ such that each $x \in f$ is an endpoint of P. In particular, if $f = \{u, v\}$, $u \neq v$, then u and v are the endpoints of P and if $f = \{x\}$, then one endpoint of P is x and the other can be any vertex of H. A set $H \subseteq V(G)$ is said to *satisfy an X-request* (f, l, X) if $\mathcal{P}^H_{f,l} \neq \emptyset$.

An M-request (f, l, M) will be simply denoted as $\varrho(f, l)$ and called *a request*.

In the next paragraphs we cover the notion of the "crucial" vertices mentioned in the earlier intuition. Roughly speaking, a vertex of F is "crucial" if the set of its descendants (vertices of its subtree) satisfies some request.

For each request $\varrho(f, l)$ we define the set $Y_{f,l} \subseteq V(\mathcal{F})$ as the set of all vertices $v \in V(\mathcal{F})$ such that $sub(v)$ satisfies the request $\varrho(f, l)$. Note that if $v \in Y_{f,l}$, then also $w \in Y_{f,l}$ for every $w \in anc(v)$, since $sub(w) \supseteq sub(v)$. Thus, if $Y_{f,l} \cap V(G'_i) \neq \emptyset$ for some i, then in particular $r_i \in Y_{f,l}$ and $Y_{f,l}$ induces a connected subtree of T_i. The request $\varrho(f, l)$ is *resolved* if the subforest $\mathcal{F}[Y_{f,l}]$ has at least $k + d + 1$ leaves.

Recall, that in the intuition we examined some d-path P in G. The resolved request basically ensures that there are at least $k + d + 1$ disjoint paths in G which satisfy said request. The idea is that the prospective solution may compromise at most k of these paths and the other parts of P may compromise at most d of these paths. As we will keep exactly $k + d + 1$ of these disjoint paths in $\widehat{G}$, we can be sure, that at least one of them will always be usable to reroute some part of P which will help us to find the desired path P' in $\widehat{G}$.

Now, we focus on the unresolved requests. Let $\mathcal{R}^*$ be the set of all requests $\varrho(f, l)$ which are *not* resolved and let $\mathcal{Y} = \bigcup_{\varrho(f,l) \in \mathcal{R}^*} Y_{f,l}$.

We are now getting to the notion of sub-requests. These have either one endpoint in M and the other in $\mathcal{Y}$, or both endpoints in $\mathcal{Y}$, or only one prescribed endpoint, which is in $\mathcal{Y}$. Note that if the two endpoint are in $\mathcal{Y}$, then, by Observation 14, either one of the endpoints is an ancestor of the other, or there is no path connecting them outside $(M \cup \mathcal{Y})$.

An $(M \cup \mathcal{Y})$-request $(g, j, M \cup \mathcal{Y})$ will be simply denoted as $\sigma(g, j)$ and called *a sub-request* if there exists $y \in \mathcal{Y}$ such that $y \in g$ and $g \subseteq M \cup anc(y)$. In particular, either $g = \{y\}$ or one of the vertices in g is y and the other vertex is in $M \cup anc(y)$.

Even though we will not formally define a *resolved sub-request*, later we will actually show that the sub-request is "resolved" if there are at least $2d$ paths satisfying it.

Description of the algorithm. We are now ready to describe the kernelization algorithm. As we mentioned earlier, the algorithm first marks some vertices and edges, which it wants to keep, and it deletes the rest of the graph. Therefore, the core of the algorithm is the marking procedure. In our case, the main procedure is called **Mark** which in turn uses a procedure called **Mark 2**. These procedures are described in Algorithm 1 and Algorithm 2, respectively.

Let us now give an insight into how the procedures **Mark** and **Mark 2** were constructed. We will start with **Mark**.

Algorithm 1 Marking procedure **Mark**.

```
1  Let M, F, and Y be as in the text.
2  Mark all the vertices and edges in G[M ∪ Y].
3  foreach resolved request ϱ(f, l) do
4      Pick arbitrary k + d + 1 leaves h_1, h_2, ..., h_{k+d+1} of F[Y_{f,l}].
5      foreach leaf h_i do
6          Pick an arbitrary path P from P_{f,l}^{sub(h_i)}.
7          Mark the vertices and edges of P.
8  foreach y ∈ Y do
9      foreach sub-request σ(g, j) such that g ⊆ M ∪ anc(y) and g ⊈ M do
10         Let C_1, C_2, ..., C_{q'} be the vertex sets of the connected components of
           G \ (M ∪ Y) such that for all i ∈ [q'], N(C_i) ∩ Y ⊆ anc(y) and C_i satisfies
           σ(g, j).
11         if q' ≥ 2d then
12             foreach i ∈ [2d] do
13                 Pick an arbitrary path P ∈ P_{g,j}^{C_i}.
14                 Mark the vertices and edges of P.
15         else
16             foreach i ∈ [q'] do
17                 Run the marking procedure Mark 2 on (σ(g, j), C_i, ∅).
```

Algorithm 2 Marking procedure **Mark 2**$\Big(\sigma(g,j), C_i, W\Big)$.

```
1  if |W| ≤ 2d and P_{g,j}^{C_i \ W} ≠ ∅ then
2      Let P ∈ P_{g,j}^{C_i \ W}.
3      Mark all the edges and vertices of P.
4      foreach v ∈ V(P) \ g do
5          W' = W ∪ {v}
6          Call Mark 2 on (σ(g, j), C_i, W')
```

The lines 3–7 deal with the resolved requests. Essentially, by preserving $k + d + 1$ corresponding paths for the request $\varrho(f, l)$, we retain all the necessary structure such that we do not create any new solutions in the reduced instance.

In the two following `for`-cycles, we first pick a vertex y of $\mathcal{Y}$. This fixes the set of ancestors $anc(y)$, i.e., it fixes the set of vertices on the path from y to the root of its tree in $\mathcal{F}$. And, for this particular y, we then pick a sub-request $\sigma(g, j)$ which lives on this fixed set $anc(y)$ and M. This allows us to look only at some components of $G \setminus (M \cup \mathcal{Y})$ and actually makes it possible for us to bound their number. The bounding happens on lines 11–14 and we can also say that the sub-request $\sigma(g, j)$ is resolved when the number of components is at least $2d$. The bound $2d$ follows from Lemma 15, which will be stated later. For a resolved sub-request we proceed similarly to resolved request. Namely, we preserve one corresponding path in each of some $2d$ of the components.

If the number of components is not large, the lines 15–17 run the second marking procedure **Mark 2** on each of these components and the aim is to bound their size.

Now, recall again, that in the intuition we examined some d-path P in G and some prospective solution S. The purpose of the marking procedure **Mark 2** is to brute-force all the possible ways of how the path P and solution S may compromise the paths which satisfy the sub-request $\sigma(g, j)$ and which are contained in the component C_i. The procedure works recursively, starts with the empty set of "compromising" vertices W and it always picks a path which was not yet compromised, marks it (so that it remains in $\widehat{G}$), and tries to compromise its vertices one by one. By doing it like this, we ensure, that all the important parts of C_i remain in $\widehat{G}$ no matter what parts of C_i were compromised.

And the main trick is that we can stop the recursion of **Mark 2** once the number of "compromising" vertices reaches $2d$. This number $2d$ again follows from Lemma 15.

Now, the kernelization can be formally summarized as follows. Run the marking procedure **Mark** on the instance (G, k). The marking results in two subsets $\widehat{V} \subseteq V(G)$ and $\widehat{E} \subseteq E(G)$ corresponding to marked vertices and edges by **Mark**. Reduce the instance (G, k) to the instance $(\widehat{G}, k)$ where $\widehat{G} = (\widehat{V}, \widehat{E})$.

With that we conclude the intuition and we continue with the formal proof of correctness. First, we state the crucial Lemma 15, then the main body of the proof follows in Lemma 16, and we finish with the proof of the size of the kernel in Lemma 17.

Lemma 15 roughly states that any reasonable solution only contains at most d vertices among each set of components considered on line 10 of Algorithm 1.

▶ **Lemma 15.** *Let (G, k) be an instance of d-PVC. Let $\widehat{G} = (\widehat{V}, \widehat{E})$ be a subgraph of G such that $\widehat{V}$ and $\widehat{E}$ are the result of running the marking procedure **Mark** on (G, k). Let S' be a solution for the instance $(\widehat{G}, k)$ of d-PVC. Let $y \in \mathcal{Y}$ and let $C_1, C_2, \ldots, C_c$ be the vertex sets of the connected components of $\widehat{G} \setminus (M \cup \mathcal{Y})$ such that $N(C_i) \cap \mathcal{Y} \subseteq anc(y)$ for $i \in [c]$. Then $\widehat{S} = (S' \setminus \bigcup_{i \in [c]} C_i) \cup anc(y)$ is a solution for $\widehat{G}$.*

Proof. If $\widehat{S}$ is a solution for $(\widehat{G}, k)$, we are done. Suppose to the contrary that it is not. Then there is a d-path P in $\widehat{G} \setminus \widehat{S}$. Assume that P is selected such that it contains the least number of vertices which are in S' i.e., $|V(P) \cap S'|$ is minimized among all d-paths in $\widehat{G} \setminus \widehat{S}$. As $S' \setminus \widehat{S} \subseteq \bigcup_{i \in [c]} C_i$, path P must contain at least one vertex from at least one set $C_i \cap S'$, because otherwise P would also be in $\widehat{G} \setminus S'$, which is a contradiction with S' being a solution for $(\widehat{G}, k)$. Further, since $N(C_i) \subseteq (M \cup \mathcal{Y})$, $N(C_i) \cap \mathcal{Y} \subseteq anc(y)$, and $anc(y) \subseteq \widehat{S}$ by assumption, we have $N(C_i) \setminus \widehat{S} \subseteq M$. Therefore $P \cap M \neq \emptyset$, as otherwise the path P would be contained in C_i, which is a contradiction with $\mathcal{M}$ being a maximal packing of d-paths.

We split the path P into segments according to the vertices of M, i.e., a *segment* of P is a sub-path $(v_1, v_2, \ldots, v_s)$ of P such that $v_2, v_3, \ldots, v_{s-1} \in V(G) \setminus M$ and either $\{v_1, v_s\} \subseteq M$ (an inner segment), or one of v_1, v_s is in M, while the other is an endpoint of P (an outer segment). The argument is the same in both cases.

Let $P' = (v_1, v_2, \ldots, v_s)$ be the segment of P which uses some vertex from $C_i \cap S'$. Observe, that the segment P' corresponds to request $\varrho(f, l) = \varrho(V(P') \cap M, s - 1)$ as $2 \leq s \leq d$ and $|V(P') \cap M| \in \{1, 2\}$. In particular, $V(P') \setminus M$ satisfies $\varrho(f, l)$.

We also know that $V(P') \cap \mathcal{Y} = \emptyset$, because $N(C_i) \setminus \widehat{S} \subseteq M$. With that we argue that the request $\varrho(f, l)$ must be resolved. Indeed, suppose it is not. By Observation 14(c) there is a vertex v in $V(P') \cap C_i$ such that that $V(P') \cap C_i \subseteq sub(v)$. But that implies that $sub(v)$ satisfies the request $\varrho(f, l)$ and, therefore, vertex v should have been included in $Y_{f,l}$ and, consequently, v should have been included in $\mathcal{Y}$, which is a contradiction with $V(P') \cap \mathcal{Y} = \emptyset$.

Now, as the request $\varrho(f,l)$ is resolved, the marking procedure **Mark** picked $k+d+1$ leaves $h_1, h_2, \ldots, h_{k+d+1}$ from $\mathcal{F}[Y_{f,l}]$ and for each such leaf h_i it marked the vertices and edges of some path $P_i \in \mathcal{P}_{f,l}^{sub(h_i)}$. Therefore, these paths $P_1, P_2, \ldots, P_{k+d+1}$ remained in $\widehat{G}$. Further, at least one of these paths is untouched by the vertices of S' and the vertices of P as $|S'| \leq k$ and $|V(P)| \leq d$, respectively. Let this one untouched path be P_i. Observe, that we can swap the segment P' with the path P_i in P to obtain a d-path P^*. But then the path P^* contains strictly fewer vertices which are in S' than P, which is a contradiction with the choice of P. ◀

Now we can prove the correctness of the algorithm.

▶ **Lemma 16** (⋆). *Let (G,k) be an instance of d-PVC. Let $\widehat{G} = (\widehat{V}, \widehat{E})$ be a subgraph of G such that $\widehat{V}$ and $\widehat{E}$ are obtained by running the marking procedure **Mark** on (G,k). Then, (G,k) is a YES instance if and only if $(\widehat{G},k)$ is a YES instance.*

Proof sketch. For the "if" direction we pick a solution S' to $\widehat{G}$ such that any application of Lemma 15 would increase its size. If S' was not a solution to G, then as in Lemma 15, we pick a special d-path P witnessing that, but this time with the least number of unmarked edges in G. Then we again split P into segments according to M and, in case the corresponding request was not resolved, further into sub-segments according to $\mathcal{Y}$. We always pick a (sub-)segment with at least one unmarked edge and we show that we can swap the (sub-)segment with some other suitable fully marked sub-path to obtain a contradiction with the choice of P. ◀

The following lemma shows the bound on the size of the kernel.

▶ **Lemma 17** (⋆). *Let (G,k) be an instance of d-PVC. Let $\widehat{G} = (\widehat{V}, \widehat{E})$ be a subgraph of G such that $\widehat{V}$ and $\widehat{E}$ are obtained by running the marking procedure **Mark** on (G,k). Then, $|\widehat{V}| = \mathcal{O}(k^4 d^{2d+9})$ and $|\widehat{E}| = \mathcal{O}(k^4 d^{2d+9})$.*

We summarize the result in the following theorem.

▶ **Theorem 18.** *d-Path Vertex Cover admits a kernel with $\mathcal{O}(k^4 d^{2d+9})$ vertices and edges, where k is the size of the solution.*

Proof. As the marking procedures **Mark** and **Mark 2** can by implemented in polynomial time, the theorem directly follows from Lemmas 16 and 17. ◀

8 Conclusion

We presented kernels with $\mathcal{O}(k^2)$ edges for 4-PVC and 5-PVC and with $\mathcal{O}(k^4 d^{2d+9})$ edges for d-PVC for any $d \geq 6$. An obvious open question is whether there is a kernel with $\mathcal{O}(k^2)$ edges for every $d \geq 6$.

Furthermore, the size of our kernel depends on d by a factor of $d^{\mathcal{O}(d)}$. We believe that this could be improved to $2^{\mathcal{O}(d)}$ with the use of representative sets. However, improving this to a factor polynomial in d would imply coNP $\subseteq$ NP/poly. As observed by Dell and Marx [9], running such a kernel with $k = 0$ would give a polynomial kernel for the d-Path problem, which would have the above mentioned implications.

Next, for 2-PVC and 3-PVC, there are kernels with linear number of vertices [19, 32]. Hence, another open question is whether such a kernel can be obtained also for say 4-PVC. Further interesting open questions can be found in the recent survey of Tu [29].

References

1 Boštjan Brešar, František Kardoš, Ján Katrenič, and Gabriel Semanišin. Minimum k-path vertex cover. *Discrete Applied Mathematics*, 159(12):1189–1195, 2011. doi:10.1016/j.dam.2011.04.008.

2 Eglantine Camby, Jean Cardinal, Mathieu Chapelle, Samuel Fiorini, and Gwenaël Joret. A primal-dual 3-approximation algorithm for hitting 4-vertex paths. In *9th International colloquium on graph theory and combinatorics*, 2014.

3 Radovan Červený and Ondřej Suchý. Faster FPT algorithm for 5-path vertex cover. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 32:1–32:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.32.

4 Radovan Červený and Ondřej Suchý. Faster FPT algorithm for 5-path vertex cover. *CoRR*, abs/1906.09213, 2019. arXiv:1906.09213.

5 Radovan Červený and Ondřej Suchý. Generating faster algorithms for d-path vertex cover. *CoRR*, abs/2111.05896, 2021. arXiv:2111.05896.

6 Maw-Shang Chang, Li-Hsuan Chen, Ling-Ju Hung, Yi-Zhi Liu, Peter Rossmanith, and Somnath Sikdar. Moderately exponential time algorithms for the maximum bounded-degree-1 set problem. *Discret. Appl. Math.*, 251:114–125, 2018. doi:10.1016/j.dam.2018.05.032.

7 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.

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

9 Holger Dell and Dániel Marx. Kernelization of packing problems. *CoRR*, abs/1812.03155, 2018. arXiv:1812.03155.

10 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 251–260. ACM, 2010. doi:10.1145/1806689.1806725.

11 Reinhard Diestel. *Graph Theory, 5th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2016.

12 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

13 Stefan Fafianie and Stefan Kratsch. A shortcut to (sun)flowers: Kernels in logarithmic space or linear time. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 – 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2015. doi:10.1007/978-3-662-48054-0_25.

14 Henning Fernau. Parameterized algorithmics for d-hitting set. *Int. J. Comput. Math.*, 87(14):3157–3174, 2010. doi:10.1080/00207160903176868.

15 Fedor V. Fomin, Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Saket Saurabh. Iterative compression and exact algorithms. *Theor. Comput. Sci.*, 411(7-9):1045–1053, 2010. doi:10.1016/j.tcs.2009.11.012.

16 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. *SIAM J. Discret. Math.*, 30(1):383–410, 2016. doi:10.1137/140997889.

17 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4), September 2016. doi:10.1145/2886094.

18 Stefan Funke, André Nusser, and Sabine Storandt. On k-path covers and their applications. *VLDB J.*, 25(1):103–123, 2016. doi:10.1007/s00778-015-0392-3.

19 Michael Lampis. A kernel of order $2k - c \log k$ for vertex cover. *Inf. Process. Lett.*, 111(23-24):1089–1091, 2011. doi:10.1016/j.ipl.2011.09.003.

20 Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. *Math. Program.*, 177(1-2):1–19, 2019. doi:10.1007/s10107-018-1255-7.

21 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

22 Marián Novotný. Design and analysis of a generalized canvas protocol. In *Proc. 4th IFIP WG 11.2 International Workshop on Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices, WISTP 2010*, pages 106–121, 2010. doi:10.1007/978-3-642-12368-9_8.

23 Elena Prieto-Rodríguez. *Systematic kernelization in FPT algorithm design.* PhD thesis, University of Newcastle, 2005. URL: http://hdl.handle.net/1959.13/1418337.

24 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.

25 Dekel Tsur. Faster deterministic parameterized algorithm for k-path. *Theor. Comput. Sci.*, 790:96–104, 2019. doi:10.1016/j.tcs.2019.04.024.

26 Dekel Tsur. l-path vertex cover is easier than l-hitting set for small l. *CoRR*, abs/1906.10523, 2019. arXiv:1906.10523.

27 Dekel Tsur. Parameterized algorithm for 3-path vertex cover. *Theor. Comput. Sci.*, 783:1–8, 2019. doi:10.1016/j.tcs.2019.03.013.

28 Dekel Tsur. An $O^*(2.619^k)$ algorithm for 4-path vertex cover. *Discret. Appl. Math.*, 291:1–14, 2021. doi:10.1016/j.dam.2020.11.019.

29 Jianhua Tu. A survey on the k-path vertex cover problem. *CoRR*, abs/2201.03397, 2022. arXiv:2201.03397.

30 Jianhua Tu and Wenli Zhou. A primal-dual approximation algorithm for the vertex cover P_3 problem. *Theor. Comput. Sci.*, 412(50):7044–7048, 2011. doi:10.1016/j.tcs.2011.09.013.

31 Mingyu Xiao and Shaowei Kou. Exact algorithms for the maximum dissociation set and minimum 3-path vertex cover problems. *Theor. Comput. Sci.*, 657:86–97, 2017. doi:10.1016/j.tcs.2016.04.043.

32 Mingyu Xiao and Shaowei Kou. Kernelization and parameterized algorithms for 3-path vertex cover. In T. V. Gopal, Gerhard Jäger, and Silvia Steila, editors, *Theory and Applications of Models of Computation – 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, volume 10185 of *Lecture Notes in Computer Science*, pages 654–668, 2017. doi:10.1007/978-3-319-55911-7_47.

33 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017. doi:10.1016/j.ic.2017.06.001.

34 Meirav Zehavi. Mixing color coding-related techniques. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms – ESA 2015 – 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 1037–1049. Springer, 2015. doi:10.1007/978-3-662-48350-3_86.