Improved Approximation for Pathwidth One Vertex Deletion and Parameterized Complexity of Its Variants

Satyabrata Jana
University of Warwick, UK

Soumen Mandal
Department of Mathematics, IIT Delhi, India

Ashutosh Rai
Department of Mathematics, IIT Delhi, India

Saket Saurabh
The Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Norway

Abstract

The pathwidth of a graph is a measure of how path-like the graph is. The Pathwidth One Vertex Deletion (POVD) problem asks whether, given an undirected graph G and an integer k, one can delete at most k vertices from G so that the remaining graph has pathwidth at most one. This is a natural variation of the classical Feedback vertex Set (FVS) problem, where the deletion of at most k vertices results in a graph of treewidth at most one. In this work, we investigate POVD in the realm of approximation algorithms. We first design a 3-approximation algorithm for POVD running in polynomial time. Then, using this constant factor approximation algorithm, we obtain a randomized parameterized approximation algorithm for POVD running in time $\mathcal{O}^*((h_\beta)^k)$, that improves the fastest existing running times for approximation ratios in the range $(1.76147, 3)$. Here the constant h_β depends on the approximation factor β alone and has value $2^{(3-\beta)}$, which lies in the range $(1, 2.3596)$, when $\beta \in (1.76147, 3)$.

Taking inspiration from two extensively studied problems, namely Connected FVS and Independent FVS, we investigate two variations of the POVD problem from the perspective of parameterized algorithms. These variations are the connected variant, called Connected pathwidth One Vertex Deletion (CPOVD) and the independent variant, called Independent Pathwidth One Vertex Deletion (IPOVD). While in CPOVD the subgraph $G[S]$ induced by the vertices to be deleted needs to be connected, in IPOVD it needs to be independent. Specifically, we show the following results.

- CPOVD can be solved in $\mathcal{O}^*(14^k)$ time and admits no polynomial kernel unless $\mathsf{NP} \subseteq \text{co-}\mathsf{NP}/\text{poly}$.
- IPOVD can be solved in $\mathcal{O}^*(7^k)$ time, and admits a kernel of size $\mathcal{O}(k^3)$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Pathwidth, Parameterized complexity, Approximation, Kernelization

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.39

Funding *Satyabrata Jana*: Supported by the Engineering and Physical Sciences Research Council (EPSRC) via the project MULTIPROCESS (grant no. EP/V044621/1).
Soumen Mandal: Supported by the Council of Scientific and Industrial Research, India (file No. 09/0086(11535)/2021-EMR-I).
Saket Saurabh: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).

1 Introduction

Treewidth is a fundamental graph parameter that measures how “tree-like” the graph is. The notion of treewidth was introduced by Robertson and Seymour in their seminal Graph Minors series [27]. Formally, it is defined via tree decompositions, where the goal is to map a graph into a tree-like structure of subsets of vertices of the graph, called *bags*, in a way that minimizes the size of the largest bag minus one. Treewidth plays a central role in the field of parameterized complexity and has been extensively studied due to its wide applicability in algorithm design, particularly for fixed-parameter tractable (FPT) algorithms. Many computationally hard problems become efficiently solvable when restricted to graphs of bounded treewidth.

A closely related parameter is *pathwidth*, which is a notion closely related to treewidth, and was also introduced by Robertson and Seymour in the Graph Minors series [26]. Here the underlying decomposition structure is restricted to paths instead of trees. While treewidth measures the tree-likeness of a graph, pathwidth measures how “path-like” the graph is.

The Feedback Vertex Set (FVS) problem is a classic graph modification problem where given a graph G and an integer k, the objective is to delete at most k vertices to obtain an acyclic graph (i.e., a forest). This problem can be viewed through the lens of treewidth, as forests are exactly the graphs with treewidth at most one. Due to this we can refer FVS problem as Treewidth One Vertex Deletion. FVS has been intensively studied in the FPT framework [12, 13, 16], leading to numerous algorithmic breakthroughs. Several variants of FVS have also been investigated. Two notable examples are: Connected Feedback Vertex Set (CFVS) [1, 6, 22] and Independent Feedback Vertex Set (IFVS)[2, 17, 21, 28], where in CFVS the subgraph of G induced by solution vertices S needs to be connected, and in IFVS it needs to be independent. Currently, the most efficient algorithm known for IFVS is provided by Li and Pilipczuk, which runs in $\mathcal{O}^*(3.619^k)$ time [17]. Regarding CFVS, Cygan et al.[7] gave the fastest known randomized algorithm for CFVS, achieving a runtime of $\mathcal{O}^*(4^k)$.

Many classical graph problems such as Vertex Cover, Feedback Vertex Set (FVS), Odd Cycle Transversal (OCT), and Dominating Set (DOM) have been extensively studied in the fields of parameterized complexity and graph algorithms. These problems are central to both theoretical research and practical applications, and their algorithmic properties – especially in the fixed-parameter tractability (FPT) setting. In recent years, increasing attention has been devoted to connected and independent variants of these problems, which often impose additional structural constraints and lead to new algorithmic challenges. Problems such as Connected Vertex Cover [14, 23], Connected DOM [11], Connected OCT [7] have emerged as natural extensions, where solutions are required to induce connected subgraphs. Similarly, Independent Dominating Set [18], Independent OCT [20], Independent s-t Separator [20], Independent Multicut [19], Independent Directed FVS [19], Independent Cutset [25] introduce independence constraints.

1.1 Pathwidth One Vertex Deletion

Analogous to FVS or Treewidth One Vertex Deletion, Philip et al.[24] introduced the Pathwidth-One Vertex Deletion (POVD) problem. In problem POVD, given a graph G and an integer k, the goal is to delete at most k vertices such that the remaining graph has pathwidth at most one. We define the problem formally as follows.

Pathwidth One Vertex Deletion (POVD)
Input: A graph $G = (V, E)$, a positive integer k.
Question: Does there exist $S \subseteq V(G)$ such that $|S| \leq k$ and pathwidth of $G - S$ is at most one?

It is known that a graph has pathwidth at most one if and only if it is a disjoint union of caterpillars [29]. A *caterpillar* is a special kind of tree where all vertices are within distance one of a central path called the *spine* [24, 29]. Hence, graphs of pathwidth at most one form a subclass of forests, exhibiting even simpler structure.

It follows from a general NP-hardness result of Lewis and Yannakakis [15] that this problem is NP-complete. Therefore, the POVD problem has attracted some attention in the parameterized complexity community and several FPT algorithms have been developed for this problem. Philip et al. [24] gave an algorithm for POVD with $\mathcal{O}^*(7^k)$ running time[1] and a kernel with $\mathcal{O}(k^4)$ vertices. Cygan et al. [8] improved these results by giving an algorithm with $\mathcal{O}^*(4.646^k)$ running time and a kernel with $\mathcal{O}(k^2)$ edges. The current fastest FPT algorithm for POVD has a running time of $\mathcal{O}^*(3.888^k)$[29]. Very recently, Esmer and Kulik [10] gave a parameterized approximation algorithm for POVD with a running time of $\mathcal{O}^*((3.888 - \varepsilon)^k)$ for some $\varepsilon > 0$, achieving approximation factors lying in the range $(1, 7)$.

1.2 Our Contributions

Following the line of research on FVS variants such as CFVS and IFVS, in this paper, we initiate the study of two natural extensions of POVD, one capturing independence, and the other emphasizing connectivity. Our research looks into these variants, revealing their intricacies with the following theme.

Connectivity vs **Independence**

Formally, we study the following connected and independent variants of POVD.

Connected Pathwidth One Vertex Deletion (CPOVD)
Input: A graph G, a positive integer k.
Question: Is there a vertex subset $S \subseteq V(G)$ of size at most k such that $G[S]$ is connected and pathwidth of $G - S$ is at most one?

Independent Pathwidth One Vertex Deletion (IPOVD)
Input: A graph G, a positive integer k.
Question: Is there a vertex subset $S \subseteq V(G)$ of size at most k such that $G[S]$ is independent and pathwidth of $G - S$ is at most one?

We obtain the following results for CPOVD and IPOVD.

- IPOVD admits a kernel of size $\mathcal{O}(k^3)$ and can be solvable in time $\mathcal{O}^*(7^k)$.
- CPOVD does not admit a polynomial kernel but can be solvable in time $\mathcal{O}^*(14^k)$.

In Section 4, we develop a kernel of size $\mathcal{O}(k^3)$ for IPOVD. Our approach builds on the framework of established kernelization algorithms for POVD [8, 24] and IFVS [21]. However, due to the inherited generality of our problem, we need to introduce substantial modifications to the reduction rules and perform a more detailed analysis. The size of our kernel matches, up to a constant factor, the best-known kernel size for IFVS. In contrast to IPOVD, in

[1] We use $\mathcal{O}^*$ notation to hide factors polynomial in the input size.

Section 5, we prove that CPOVD does not admit a polynomial kernel parameterized by the solution size unless $\mathsf{NP} \subseteq \text{co-}\mathsf{NP}/\text{poly}$. This negative result follows via a straightforward reduction from the CONNECTED VERTEX COVER problem, which is also known not to have a polynomial kernel under the same assumption.

On the algorithmic side, in Section 6, we give an FPT algorithm for CPOVD running in time $\mathcal{O}^*(14^k)$. The core idea is to first eliminate certain obstructive structures from the graph through branching, simplifying the instance. Taking advantage of this simplified graph, we then invoke an FPT algorithm for GROUP STEINER TREE in a black-box fashion with carefully constructed input, allowing us to test whether any minimal POVD solution can be extended to a connected set that meets the size constraint. Beyond the $\mathcal{O}(k^3)$ kernel for IPOVD, in Section 7 we also present an FPT algorithm for IPOVD running in time $7^k \cdot k^{\mathcal{O}(1)} + n^{\mathcal{O}(1)}$. This algorithm is conceptually similar to the earlier FPT approach for CPOVD: we first branch to remove some of the obstructive structures, and then, again with the help of the simplified structure of the residual graph, check whether any minimal POVD solution of size at most k is independent.

Interestingly, despite the active study of approximation algorithms for other vertex deletion problems, no polynomial-time approximation algorithm better than the trivial 7-approximation was previously known for POVD. We provide a 3-approximation algorithm for POVD running in polynomial time. This algorithm is based on two key observations that any pathwidth-one vertex deletion set also forms a feedback vertex set and, POVD can be solved in polynomial time when the input graph is acyclic.

Furthermore, we investigate POVD in the realm of parameterized approximation. Recently, Esmer and Kulik [10], presented a parameterized approximation algorithm for POVD that achieve an approximation guarantee with factors in the range $(1, 7)$ in randomized FPT running time, that improves over the best known parameterized algorithm for POVD. In Section 3.3, with the help of the 3-approximation algorithm, we obtain another parameterized approximation algorithm that improves upon the existing randomized FPT running times of algorithms given by [10] for approximation ratios in the range $(1.76147, 3)$. Due to space constraints, some proofs have been moved to Appendix (Appendix A).

2 Preliminaries

All the graphs we consider are simple, undirected and unweighted graphs. For a graph G, we denote the set of vertices of the graph by $V(G)$ and the set of edges of the graph by $E(G)$. We denote $|V(G)|$ by n, where the graph is clear from the context. An edge with endpoint u and v is denoted by uv. For a set $S \subseteq V(G)$, the subgraph of G *induced* by S is denoted by $G[S]$ and it is defined as the subgraph of G with vertex set S and edge set $\{uv \in E(G) : u, v \in S\}$. We denote the subgraph $G[V(G) \setminus S]$ as $G - S$. For any graph $G = (V, E)$ and $v \in V$, a vertex $u \in V$ is said to be a neighbour of v if $vu \in E(G)$. The open neighbourhood of v is denoted by $N(v)$ and it is defined as $N(v) = \{u \in V : vu \in E(G)\}$. We denote degree of a vertex v by $d(v)$ where $\deg(v) = |N(v)|$. The closed neighbourhood of a vertex v is $N[v] = N(v) \cup \{v\}$. By $\Delta(G)$, we denote the maximum degree of a vertex in G. We denote a polynomial function of n as $\text{poly}(n)$. A set $S \subseteq V$ is a *pathwidth-one vertex deletion set* (povd) if $G - S$ has pathwidth at most one. If $G[S]$ is an independent set, then S is an independent povd (ipovd). If $G[S]$ is connected, then S is a connected povd (cpovd).

Parameterized Complexity. We refer to [5] for an introduction to the area. A parameterized problem (x, k) where x is a string over Σ and a parameter $k \in \mathbb{N}$ is fixed-parameter tractable (or FPT) if it can be solved in $f(k) \cdot |x|^{\mathcal{O}(1)}$ time for some computable function f. A kernelization algorithm, or in short, *kernelization*, for a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}$ is an algorithm that, given $(X, k) \in \Gamma^* \times \mathbb{N}$, outputs in time polynomial in $|X| + k$, a pair $(X', k') \in \Gamma^* \times \mathbb{N}$ such that (a) $(X, k) \in \Pi$ if and only if $(X', k') \in \Pi$ and (b) $|x'|, |k| \leq g(k)$, where g is some computable function depending only on k. The output of kernelization (X', k') is referred to as the kernel and the function g is referred to as the size of the kernel. If $g(k) \in k^{\mathcal{O}(1)}$, then we say that Π admits a polynomial kernel.

Branching Algorithm. We need to define the notion of a branching tree for the analysis of our algorithms. An algorithm or a procedure $\mathcal{A}$ on an instance I of a problem is called a branching algorithm if $\mathcal{A}$ can potentially recursively call itself on instances $I_1, I_2, \ldots, I_t$ for $t \geq 2$ such that output of $\mathcal{A}$ depends on the output of the recursive calls. A run of the algorithm on an instance I is said to have executed a branching step, if it recursively calls itself on at least two instances. A branching tree associated with an algorithm $\mathcal{A}$ and an input instance I is a tree where the root node of the tree is associated with the instance I. If on an instance I, branching step is executed and recursive call is made on the instances $I_1, I_2, \ldots, I_t$, then the node associated with I in the branching tree would have t children, each associated with one of the instances on which the recursive call was made. If there is no branching step executed on any of the instances associated with a node in the branching tree, then that node would be a leaf node in the branching tree.

Now, we state some useful known results on graphs with pathwidth at most one.

▶ **Fact 1** ([29, Lemma 1]). *Let G be a graph. The following assertions are equivalent.*

- *The pathwidth of G is at most 1.*
- *G is a caterpillar forest.*
- *G is acyclic and does not contain a subgraph isomorphic to T_2 (see Figure 1).*

▶ **Fact 2** ([24, Lemma 3]). *Let $\mathcal{S} = \{T_2, K_3, C_4\}$, where C_4 is a cycle of length 4. If G is a graph that does not contain any element of $\mathcal{S}$ as a subgraph, then each connected component of G is either a tree or a cycle with zero or more pendant vertices ("hairs") attached to it.*

3 Approximation for POVD

In this section, we describe the approximation algorithms for Pathwidth One Vertex Deletion. We first obtain a 3-approximation algorithm for POVD that runs in polynomial time. Then using this polynomial time approximation and results of [10], we obtain parameterized β-approximation for every $\beta \in [1, 3]$, the running time for which scales between the running times of the best known FPT algorithm for POVD ($\beta = 1$) and the polynomial time 3-approximation algorithm designed in Section 3.2 ($\beta = 3$). Before introducing these approximation techniques, we begin by proving a helpful result: POVD can be solved in polynomial time on acyclic graphs.

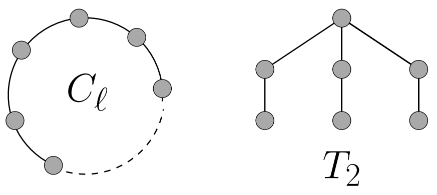

Figure 1 Forbidden subgraphs in graphs of pathwidth at most one. C_ℓ denotes a cycle of size ℓ.

3.1 Polynomial Time Algorithm Solving POVD on Acyclic Graphs

We first solve POVD on trees using a simple greedy approach. By Fact 1, the obstructions are cycles and the graph T_2. Since the input is a tree, the only possible obstruction is T_2, which has seven vertices, see Figure 1.

Given a tree H, we first root it at an arbitrary vertex $r \in V(H)$ using BFS. We then process the tree in a bottom-up manner to detect occurrences of T_2. Whenever a copy of T_2 is encountered for the first time in this traversal, we select a vertex v in the highest layer (closest to the root) of that copy. We include v in the solution set and remove v together with the entire subtree rooted at v from H. The process is then repeated: we again scan the tree bottom-up, locate the next copy of T_2, pick a highest-layered vertex v, and remove its subtree.

The algorithm terminates once the tree contains no copy of T_2. At that point, the set of selected vertices is returned as the solution.

We now argue the correctness of this approach.

▷ Claim 1. The above algorithm returns a minimum pathwidth-one vertex deletion set when the input graph is a tree.

Proof. Let H be a tree and let A be a copy of T_2 in H such that our algorithm identifies A during its first execution and selects one of the topmost vertices v from A into the solution. To prove correctness, it suffices to show that for any other T_2, say B in H intersecting A, the vertex v lies in $V(A) \cap V(B)$. If this holds, then choosing v is always as good as choosing any other vertex from A.

Assume for contradiction that $v \notin V(B)$, while $V(A) \cap V(B) \neq \emptyset$. Then there must exist a vertex $u \in V(A) \cap V(B)$ with $u \neq v$. Since v is chosen as one of the topmost vertices of A, the vertex u must occur at the same level or lower than v in the rooted tree. This would imply that there exist two distinct root-to-u paths in H: one passing through v and another through B that avoids v. Consequently, this would create a cycle in H, contradicting the assumption that H is a tree. Hence, v must belong to every T_2 intersecting A, establishing the claim. ◁

Since BFS runs in polynomial time and each scan for T_2 also takes polynomial time, the overall running time of the greedy algorithm is polynomial. Thus, POVD can be solved in polynomial time on trees. Moreover, since every connected component of an acyclic graph is itself a tree, we can apply the same approach independently on each component of an acyclic graph. This yields the following theorem.

▶ **Theorem 2.** POVD *can be solved in polynomial time on acyclic graphs.*

3.2 Polynomial Time 3-approximation for POVD

In this section, we prove the following result.

▶ **Theorem 3.** *There exists a polynomial time* 3*-approximation algorithm for* POVD.

The approach for our approximation algorithm is based on two key observations that any pathwidth-one vertex deletion set also forms a feedback vertex set and, as shown in the previous subsection, POVD can be solved in polynomial time when the input graph is acyclic. Our approximation algorithm proceeds as follows.

- Given a graph G, we first compute a 2-approximate feedback vertex set S_1, using known 2-approximation algorithms for the FEEDBACK VERTEX SET problem [3, 4]. Removing S_1 from G yields a graph $G - S_1$ that is acyclic.

- Next, we compute the minimum pathwidth-one vertex deletion set for $G - S_1$ using the polynomial-time algorithm described in Section 3.1. Let this set be S_2. Finally, we return the set $S = S_1 \cup S_2$.

Since S_2 is a pathwidth-one vertex deletion for $G - S_1$, $S = S_1 \cup S_2$ must be a pathwidth-one vertex deletion for G. Also, the above-described algorithm runs in polynomial time, as S_1 can be found in polynomial time using one of the 2-approximation algorithms for Feedback Vertex Set, and for S_2, we again need polynomial time only because of the result in Section 3.1. We next argue for the size bound of $S = S_1 \cup S_2$ in the following claim.

$\triangleright$ Claim 4. For any given graph G, the size of S, returned by the algorithm described above, can be at most three times the size of an optimal pathwidth-one vertex deletion set of G.

Proof. Let S' be an optimal pathwidth-one vertex deletion for the graph G. We want to prove that $|S| \leq 3|S'|$. We know that $G - S'$ is a caterpillar forest. Also, let F be an optimal fvs for G. Since $G - S'$ is also acyclic, we must have $|F| \leq |S'|$. Again, since S_1 is a 2-approximate fvs solution, we get $|S_1| \leq 2|F| \leq 2|S'|$. On the other side, S_2 is an optimal pathwidth-one vertex deletion for $G - S_1$, which implies $|S_2| \leq |S'|$. Combining these, we get, $|S| = |S_1| + |S_2| \leq 2|S'| + |S'| \leq 3|S'|$. $\triangleleft$

3.3 Parameterized Approximation for POVD

In this subsection, we improve the running time of the parameterized approximation algorithm for POVD introduced by Esmer and Kulik [10], for approximation factors between 1.76147 and 3. Recall that in [10], the authors provided running time for all approximation factors in the range from 1 to 7. However, due to our 3-approximation described in the previous subsection, it is no longer necessary to consider approximation factors greater than 3 in the parameterized setting for POVD. By combining our 3-approximation with the "Sampling with a black box" technique given in [10], we obtain faster β-approximation algorithms for all β such that $\beta \in (1.76147, 3]$.

Esmer and Kulik [10] showed that a β-approximation for a vertex deletion minimization problem π can be obtained in time $\mathcal{O}^*((\text{convert}(\alpha, \beta, c, q))^k)$ time, where the precondition is an α-approximation algorithm for the problem π that runs in time $\mathcal{O}^*(c^k)$ with $c \leq \exp\left(\beta \cdot \mathcal{D}\left(\frac{1}{\beta} \| q\right)\right)$, and that π has a sampling step where success probability of each step is at least q. Here $\mathcal{D}(\cdot\|\cdot)$ denotes Kullback-Leibler divergence[2]. The formula of $\text{convert}(\alpha, \beta, c, q)$ is as follows.

$$\text{convert}(\alpha, \beta, c, q) := \begin{cases} c \cdot \exp\left(\dfrac{\delta^*_{\text{right}} \cdot \mathcal{D}\left(\frac{1}{\delta^*_{\text{right}}} \| q\right) - \ln(c)}{\delta^*_{\text{right}} - \alpha} \cdot (\beta - \alpha) \right) & \text{if } \delta^*_{\text{right}}(\alpha, c, q) \geq \beta \geq \alpha; \\ c \cdot \exp\left(\dfrac{\delta^*_{\text{left}} \cdot \mathcal{D}\left(\frac{1}{\delta^*_{\text{left}}} \| q\right) - \ln(c)}{\delta^*_{\text{left}} - \alpha} \cdot (\beta - \alpha) \right) & \text{if } \delta^*_{\text{left}}(\alpha, c, q) \leq \beta \leq \alpha; \\ \exp\left(\beta \cdot \mathcal{D}\left(\frac{1}{\beta} \| q\right)\right) & \text{otherwise.} \end{cases} \tag{1}$$

[2] Kullback-Leibler Divergence: given two numbers $a, b \in [0, 1]$, the Kullback-Leibler divergence of a and b is defined to be

$$\mathcal{D}(a\|b) = a \cdot \ln\left(\frac{a}{b}\right) + (1 - a) \cdot \ln\left(\frac{1-a}{1-b}\right)$$

Here, $\delta^*_{\text{left}}(\alpha, c, q) \in (1, \alpha]$ and $\delta^*_{\text{right}}(\alpha, c, q) \in \left[\alpha, \frac{1}{q}\right]$ are defined to be the unique numbers which satisfy

$$\mathcal{D}\left(\frac{1}{\alpha}\Big\|\frac{1}{\delta^*_{\text{left}}(\alpha, c, q)}\right) = \mathcal{D}\left(\frac{1}{\alpha}\Big\|\frac{1}{\delta^*_{\text{right}}(\alpha, c, q)}\right) = \mathcal{D}\left(\frac{1}{\alpha}\Big\|q\right) - \frac{\ln(c)}{\alpha}. \tag{2}$$

Note that for POVD, it was previously known that an α-approximation with $\alpha = 1$ (which is same as an exact algorithm) can be obtained in time $\mathcal{O}^*(3.888^k)$ [29], and there is also a sampling step for POVD that succeeds with probability $q = \frac{1}{7}$ [10]. Building on this precondition, a β-approximation has been developed in [10], which runs in time $\mathcal{O}^*\left((d_\beta)^k\right)$ for every $1 \le \beta \le 7$, where

$$d_\beta = \text{convert}(1, \beta, 3.888, \frac{1}{7}) = \begin{cases} 3.888 \cdot (0.519)^{\beta-1} & \text{if } 1 \le \beta \le 1.8; \\ \exp\left(\beta \cdot \mathcal{D}\left(\frac{1}{\beta}\|\frac{1}{7}\right)\right) & \text{if } 1.8 < \beta \le 7. \end{cases} \tag{3}$$

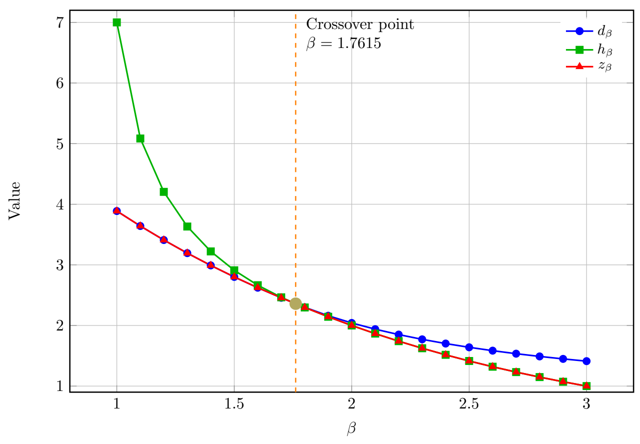

Figure 2 Comparison between the bases of the running times: d_β corresponding to the β-approximation from [10], and z_β corresponding to the β-approximation presented in Theorem 5.

Now, because of our new α-approximation with $\alpha = 3$ that runs in time $\mathcal{O}^*(c^k)$ with $c = 1$ (since it is a polynomial time 3-approximation), from Equation (1), we can obtain another parameterized β-approximation for every $\beta \in [1, 3]$, running in time $\mathcal{O}^*\left((h_\beta)^k\right)$, where

$$h_\beta = \text{convert}\left(3, \beta, 1, \frac{1}{7}\right).$$

Now, we compute convert $\left(3, \beta, 1, \frac{1}{7}\right)$. Note that, to evaluate convert $\left(3, \beta, 1, \frac{1}{7}\right)$ for $\beta \le 3$, we need the value of $\delta^*_{\text{left}}(3, 1, \frac{1}{7})$. Let $x = \delta^*_{\text{left}}(3, 1, \frac{1}{7})$, then from Equation (2), we have

$$\mathcal{D}\left(\frac{1}{3}\Big\|\frac{1}{x}\right) = \mathcal{D}\left(\frac{1}{3}\Big\|\frac{1}{7}\right) - \frac{\ln(1)}{3}.$$

This implies,

$$\frac{1}{3} \cdot \ln\left(\frac{x}{3}\right) + \frac{2}{3} \cdot \ln\left(\frac{2x}{3(x-1)}\right) = \frac{1}{3} \cdot \ln\left(\frac{7}{3}\right) + \frac{2}{3} \cdot \ln\left(\frac{14}{18}\right).$$

Further simplifying, we get

$$4x^3 = \frac{7^3 \cdot (x-1)^2}{9} \implies x = \frac{7}{4}, \text{ as } x = \delta^*_{\text{left}}(3, 1, \frac{1}{7}) \in (1, 3].$$

Now, using $\delta^*_{\text{left}}\ (3, 1, \frac{1}{7}) = \frac{7}{4}$ in Equation (1), we get

$$h_\beta = \text{convert}\left(3, \beta, 1, \frac{1}{7}\right) = \begin{cases} \exp\left(\frac{\frac{7}{4}\cdot\mathcal{D}\left(\frac{1}{\frac{7}{4}}\|\frac{1}{7}\right)}{\frac{7}{4}-3}\cdot(\beta-3)\right) & \text{if } 1.75 \le \beta \le 3; \\ \exp\left(\beta\cdot\mathcal{D}\left(\frac{1}{\beta}\|\frac{1}{7}\right)\right) & \text{otherwise.} \end{cases}$$

This gives

$$h_\beta = \text{convert}\left(3, \beta, 1, \frac{1}{7}\right) = \begin{cases} 2^{(3-\beta)} & \text{if } 1.75 \le \beta \le 3; \\ \exp\left(\beta\cdot\mathcal{D}\left(\frac{1}{\beta}\|\frac{1}{7}\right)\right) & \text{otherwise.} \end{cases} \tag{4}$$

Thus, we now have two different parameterized β-approximation algorithms for POVD, running in time $\mathcal{O}^*((d_\beta)^k)$ and $\mathcal{O}^*((h_\beta)^k)$, respectively. By comparing d_β and h_β, we observe that $d_\beta < h_\beta$ when $1 \le \beta < 1.76147$, while $h_\beta < d_\beta$ when $1.76147 < \beta \le 3$.

Combining these observations, we obtain the following theorem.

▶ **Theorem 5.** *For each $1 \le \beta \le 3$, POVD has a β-approximation algorithm running in time $\mathcal{O}^*((z_\beta)^k)$, where*

$$z_\beta = \begin{cases} 3.888\cdot(0.519)^{(\beta-1)} & \text{if } 1 \le \beta \le 1.76147; \\ 2^{(3-\beta)} & \text{if } 1.76147 < \beta \le 3. \end{cases}$$

4 Kernelization for Independent POVD

In this section, we present a polynomial kernel of size $\mathcal{O}(k^3)$ for IPOVD. Given an instance (G, k), our kernelization procedure first reduces it to an equivalent instance of a more general problem called EXT-IPOVD. In EXT-IPOVD, the input is a triple (G, R, k), and the goal is to determine whether there exists an independent pathwidth-one vertex deletion set in G that is disjoint from the set R. For any instance (G, k) of IPOVD, we can create an equivalent instance of EXT-IPOVD by simply setting $R = \emptyset$. Note that when $k \le 3$, we can solve an input instance (G, k) of IPOVD in polynomial time by enumerating all possible subsets of size at most k. Thus, for a given instance (G, k), we assume that $k > 3$.

Table 1 Values of d_β (from [10]), h_β (from Equation (4)) and z_β (from Theorem 5) for β ranging from 1.0 to 3.0 in increments of 0.1. Highlighted row shows the point where z_β starts outperforming d_β.

β	d_β	h_β	z_β	β	d_β	h_β	z_β
1.0	3.888	7.0000	3.888	2.0	2.0417	2.0000	2.0000
1.1	3.6412	5.0846	3.6412	2.1	1.9391	1.8660	1.8660
1.2	3.4100	4.2041	3.4100	2.2	1.8498	1.7411	1.7411
1.3	3.1936	3.6324	3.1936	2.3	1.7713	1.6245	1.6245
1.4	2.9908	3.2220	2.9908	2.4	1.7018	1.5157	1.5157
1.5	2.8010	2.9102	2.8010	2.5	1.6399	1.4142	1.4142
1.6	2.6232	2.6642	2.6232	2.6	1.5844	1.3195	1.3195
1.7	2.4567	2.4647	2.4567	2.7	1.5346	1.2311	1.2311
1.7615	2.3596	2.3596	2.3596	2.8	1.4896	1.1486	1.1486
1.8	2.3007	2.2973	2.2973	2.9	1.4487	1.0717	1.0717
1.9	2.1604	2.1435	2.1435	3.0	1.4115	1.0000	1.0000

We then design a kernel of size $\mathcal{O}(k^3)$ for Ext-IPOVD. Once we obtain that we then convert it back into an equivalent instance of IPOVD of size $\mathcal{O}(k^3)$. The following lemma formalizes this transformation.

▶ **Lemma 6.** *Given an instance (G, R, k) of size $\mathcal{O}(k^3)$ for* Ext-IPOVD*, we can construct an equivalent instance $(H, k+1)$ for* IPOVD *of size $\mathcal{O}(k^3)$.*

Proof. Let (G, R, k) be an instance of Ext-IPOVD with size $\mathcal{O}(k^3)$. From (G, R, k), we create an equivalent instance $(H, k+1)$ of IPOVD as follows. We define the vertex set $V(H)$ by extending $V(G)$ with $2k+5$ additional vertices $\{u\} \cup \{x_1, \ldots, x_{k+2}\} \cup \{y_1, \ldots, y_{k+2}\}$. The edge set $E(H)$ is formed the following way. We extend the edge set $E(H)$ by adding the the following edges. For each $i \in [k+2]$, we add the edges $\{x_iy_i, ux_i, uy_i\}$, and then add an edge ur for every $r \in R$.

Observe that any ipovd solution Q for H of size at most $k+1$ must include the vertex u. Otherwise, Q would need to include at least one vertex from each of the $k+2$ disjoint pairs $\{x_i, y_i\}$, contradicting the size bound of $k+1$. Since Q is an independent set that includes u, it cannot contain any vertex from R. Therefore, $Q \setminus \{u\}$ forms an independent set in G of size at most k, disjoint from R. Furthermore, as G is an induced subgraph of $H \setminus \{u\}$, $Q \setminus \{u\}$ is also a valid povd solution for G. Conversely, let Q' be an ipovd solution of size at most k for the instance (G, R, k). Since Q' avoids all vertices in R, the set $Q' \cup \{u\}$ forms a valid ipovd solution for H of size at most $k+1$. Finally, since the size of G is bounded by $\mathcal{O}(k^3)$ and we only introduce $\mathcal{O}(k)$ new vertices and $\mathcal{O}(k^3)$ additional edges, the total size of H remains bounded by $\mathcal{O}(k^3)$. ◀

So now we turn our attention to kernelization for Ext-IPOVD. We develop several reduction rules for this purpose and apply them exhaustively to any given instance (G, R, k) of Ext-IPOVD. When multiple reduction rules are applicable simultaneously, we always choose the one with the smallest index. Once no further reduction rules apply, we show that the size of the resulting instance is bounded by $\mathcal{O}(k^3)$.

Note that certain reduction rules can produce trivial `Yes` instances (the instance $(G = K_2, R = \emptyset, k = 0)$) or trivial `No` instances (the instance $(G = K_3, R = \emptyset, k = 0)$) for Ext-IPOVD. Some of our rules are slight adaptations of known reductions, while others are specifically tailored to this problem. Below, we present these rules and provide proofs of correctness for all non-trivial rules. We begin by describing three simple reduction rules.

▶ **Reduction Rule 1.** *If $k \leq 0$ and G has pathwidth at least two, return a trivial* `No` *instance.*

▶ **Reduction Rule 2.** *If $k \geq 0$ and G has pathwidth at most one, return a trivial* `Yes` *instance.*

▶ **Reduction Rule 3.** *If G has a connected component C such that C is caterpillar, remove C from the graph. Return the instance $(G - V(C), R \setminus V(C), k)$.*

The proof of the correctness of the above three reduction rules is straightforward. Below we present few other rules (Reduction Rules 4-8), correctness for each reduction rule, are provided in the Appendix (Section A.1).

▶ **Reduction Rule 4.** *Given an instance (G, R, k), if a vertex $v \in V(G)$ has more than $k+1$ pendant neighbors, retain any $k+1$ of them, ensuring that at least one is from R if v has any pendant neighbor in R. Let D_v be the set of deleted pendant neighbors, G' the resulting graph, and update $R' = R \setminus D_v$. Return the instance (G', R', k).*

Now, we will state two known reduction rules given in [24] with slight modification.

▶ **Reduction Rule 5.** *Let u be a vertex in G such that there exists a matching M of size $k+3$ in the graph $G-u$, where each edge in M has at least one endpoint in the neighborhood $N(u)$. If $u \in R$, then return trivial* `No` *instance. Otherwise, remove u from the graph, add all vertices in $N(u)$ to the set R, and decrease k by one. Return the instance $(G-u,\ R\cup N(u),\ k-1)$.*

▶ **Reduction Rule 6.** *Let $v_0, v_1, v_2, \ldots, v_{p+1}$ be a path in G such that, for every vertex v_i with $1 \le i \le p$, all its neighbors other than v_{i-1} and v_{i+1} are pendant vertices. If $p \ge 7$, contract the edge (v_3, v_4) in G, and assign the resulting vertex v_3^4 in R if both v_3 and v_4 are in R. Denote the resulting graph by G'. The new instance is $\big(G',\ R',\ k\big)$, where*

$$R' = \begin{cases} \big(R \setminus \{v_3, v_4\}\big) \cup \{v_3^4\}, & \text{if } v_3^4 \in R, \\ R \setminus \{v_3, v_4\}, & \text{otherwise.} \end{cases}$$

Towards proving the correctness of Reduction Rule 6, we use the following observation.

▶ **Observation 7** ([24, Observation 2]).
1. *Let G be a graph that contains at least one cycle. Any graph G' obtained from G by contracting an edge of G also contains at least one cycle (possibly with parallel edges).*
2. *Let G be a graph that contains a T_2 as a subgraph. If G' is a graph obtained from G by contracting an edge uv where either u or v (or both) is not part of any T_2 in G, then G' also contains a T_2 as a subgraph.*

Let v be a vertex in a graph G, and let $\ell \in \mathbb{N}$. An *ℓ-flower* passing through v is a set of ℓ distinct cycles in G such that each cycle contains v and no two cycles share any vertex other than v. The vertex v is said to be at the center of the flower.

▶ **Reduction Rule 7.** *Let u be a vertex of the graph G such that u is the center of a $(k+1)$-flower. If $u \in R$, then return trivial* `No` *instance. Otherwise, remove u from the graph, add all vertices in $N(u)$ to the set R, and decrease k by one. Return the instance $\big(G[V(G)\setminus\{u\}],\ R\cup N(u),\ k-1\big)$.*

▶ **Lemma 8.** *Let $\alpha = \max\{5, k+2\}$. Consider an* Ext-IPOVD *instance (G,R,k). Assume that there is a vertex $u \in V(G)$ of degree at least $(4k+7)+\alpha(k+2)$ and Reduction Rules 1-7 cannot be applied. Then one can, in polynomial time, find disjoint sets of vertices $X', Y' \subseteq V(G)\setminus\{u\}$ and a function $\phi : X' \to \binom{Y'}{\alpha}$ satisfying the following properties:*
1. *every vertex in Y' is connected by a single edge to u,*
2. *Y' is an independent set in G,*
3. *$N(Y') = X' \cup \{u\}$, and each vertex from Y' has a neighbor in X',*
4. *the sets $\phi(x) \subseteq N(x)$ are disjoint for any two different $x \in X'$ (as before we denote the elements of $\phi(x)$ by $y_1^x, y_2^x, \ldots, y_\alpha^x$ in Y' and call them the private neighbors of x).*

The above mentioned lemma is similar to Lemma 7 in [8]. The only difference lies in the bound on $\deg(u)$. Since our graph is simple, the maximum number of non-pendant neighbors of u can be at most $k+1$, whereas in their case it is $2k+1$. The proof of our Lemma 8 follows directly from the proof of Lemma 7 in [8].

▶ **Reduction Rule 8.** *Let u be a vertex of degree at least $(4k+7)+\alpha(k+2)$, and assume that none of the earlier reduction rules are applicable. Let X' and Y' be the nonempty sets obtained by applying Lemma 8. Construct the graph G' by deleting all but two edges between u and the vertices y_i^x, for each $x \in X'$. Then, put all vertices in Y' to R. Return the instance $\big(G',\ R\cup Y',\ k\big)$.*

If the kernelization algorithm for Ext-IPOVD returns a trivial `Yes` or trivial `No` instance, it is straightforward to observe that the size of such instances is constant. Therefore, we now focus on the case where the kernelization produces a non-trivial instance (G', R', k'). Assuming the original input instance (G, R, k) is a `Yes`-instance, we will argue that the size of G' is bounded by $\mathcal{O}(k^3)$. By correctness of the reduction rules, we have that (G, R, k) is a `Yes`-instance if and only if (G', R', k') is a `Yes`-instance. Assume that (G', R', k') is a `Yes`-instance and let S be a solution of size at most k'. Since no reduction rule increases the value of k, it follows that $k' \le k$. We now partition the set $V^\star = V(G') \setminus S$ into the following subsets.

$$V_1 = \Big\{v \in V^\star : \deg_{G'}(v) = 1, N(v) \cap S \neq \emptyset\Big\}; \quad V_2 = \Big\{v \in V^\star : \deg_{G'}(v) = 1, N(v) \cap S = \emptyset\Big\};$$

$$V_3 = \Big\{v \in V^\star : \deg_{G'-S}(v) = 1, N(v) \cap S \neq \emptyset\Big\}; \; V_4 = \Big\{v \in V^\star : \deg_{G'-S}(v) > 2, N(v) \cap S \neq \emptyset\Big\};$$

$$V_5 = \Big\{v \in V^\star : \deg_{G'-S}(v) > 2, N(v) \cap S = \emptyset, \exists u \in N(v) : u \in V_3\Big\};$$

$$V_6 = \Big\{v \in V^\star : \deg_{G'-S}(v) > 2, v \notin V_4 \cup V_5\Big\}; \; V_7 = \Big\{v \in V^\star : \deg_{G'-S}(v) = 2, N(v) \cap S \neq \emptyset\Big\};$$

$$V_8 = \Big\{v \in V^\star : \deg_{G'-S}(v) = 2, N(v) \cap S = \emptyset\Big\}.$$

▶ **Lemma 9.** *If the input instance of* Ext-IPOVD *is `Yes` instance, then* $|V(G')| \le 61k^3 + 319k^2 + 414k + 42$*, and* $|E(G')| \le 62k^3 + 327.5k^2 + 423.5k + 42$.

Proof. We assume that none of the reduction rules are applicable to the instance (G', R', k'). We now proceed to bound the size of each V_i individually.

Vertices in V_1 have exactly one neighbor, which must lie in S. Since $|S| \le k$ and, by Reduction Rule 4, each vertex in S can have at most $k + 1$ such pendant neighbors, the size of V_1 is bounded by $k(k + 1) = k^2 + k$.

Next, observe that if $v \in V_7$ or $v \in V_4$, then v must lie on the spine of a caterpillar in the subgraph $G' - S$. Suppose there are i vertices from V_7 in a caterpillar. Then, in the subgraph induced by these i vertices and their neighbors, there exists a matching of size at least $\lfloor \frac{i}{2} \rfloor$. Now consider a vertex $u \in S$ that is adjacent to more than $3k + 9$ vertices of V_7. In the graph induced by these neighbors and their adjacent vertices, we can obtain a matching of size at least $k + 3$, where each matching edge has at least one endpoint adjacent to u. This would trigger Reduction Rule 5, contradicting our assumption. Therefore, each vertex in S can be adjacent to at most $3k + 9$ vertices of V_7. By the same reasoning, each vertex in S can also be adjacent to at most $3k + 9$ vertices of V_4. Hence, when Reduction Rule 5 is not applicable, the total number of vertices in each of V_7 and V_4 is bounded by $k(3k + 9) = 3k^2 + 9k$.

Notice that, for every $v \in V_5$, there is a private vertex $u \in V_3$ such that uv is an edge. Thus, if there are x such vertices $u \in V_3$ adjacent to vertices in V_5, then $|V_5| = x$. Moreover, in the subgraph of G' induced by these u and their neighbors in $V(G') \setminus S$, there exists a matching of size x. Consequently, if there is a vertex $w \in S$ adjacent to $k + 3$ such vertices $u \in V_3$ with neighbors in V_5, then Reduction Rule 5 would be applicable. Therefore, for each vertex in S, there can be at most $k + 2$ such vertices u, implying that $|V_5| \le k(k + 2)$.

Observe that the vertices of V_6 and V_8 lie on the spine of a caterpillar in $G' - S$, such that neither they nor their pendant neighbors are adjacent to S. Since Reduction Rule 6 is no longer applicable, in any spine of a caterpillar in $G' - S$, there cannot be more than nine consecutive vertices from $V_6 \cup V_8$. Additionally, because Reduction Rule 6 is no longer applicable, there must be at least one vertex from V_4, V_5, or V_7 after any sequence of at most nine consecutive vertices from $V_6 \cup V_8$ in such a spine. Therefore, $|V_6 \cup V_8| \le 9(7k^2 + 20k)$.

Note that the neighbors of any vertex in V_2 belong to some V_i with $i \geq 3$. Let $v \in V_2$ be adjacent to some $u \in V_3$. As before, there can be at most $k+2$ such vertices u (otherwise Reduction Rule 5 would be applicable). For any vertex in V_i with $i \geq 4$, there can be at most $k+1$ adjacent vertices in V_2 by Reduction Rule 4. Therefore, in total, the size of V_2 is bounded by $|V_2| \leq (k+2)+(k+1)\big(|V_4|+|V_5|+|V_6|+|V_7|+|V_8|\big) = 70k^3+270k^2+201k+2$.

Furthermore, if none of the reduction rules are applicable, then for any $u \in V(G')$, we have: $\deg(u) \leq (4k+7)+\alpha(k+2)$, where $\alpha = \max\{5, k+2\}$. Therefore, the total number of vertices adjacent to S is bounded by $(4k^2+7k)+\alpha(k^2+2k)$. Hence, $|V_3| \leq (4k^2+7k)+\alpha(k^2+2k) = k^3+8k^2+11k$ (note that according to our initial assumption, $\alpha = k+2$).

Hence, finally we get

$$|V(G')| = |S| + \sum_{i=1}^{8} |V_i| \leq 71k^3+349k^2+254k+2.$$

Similarly, the number of edges with one endpoint in S and the other in $V(G') \setminus S$ is at most $(4k^2+7k)+\alpha(k^2+2k) = (4k^2+7k)+(k+2)(k^2+2k)$. Since $G'-S$ is a caterpillar forest, the total number of edges in $G'-S$ is at most $|V(G') \setminus S|-1$. Finally, the number of edges in $G'[S]$ is bounded by $\frac{k(k-1)}{2}$. Therefore, the total number of edges in G' is at most $\Big(|V(G') \setminus S|-1\Big) + \Big[\text{number of edges between } S \text{ and } V(G') \setminus S\Big] + \frac{k(k-1)}{2} = (71k^3+349k^2+253k+1)+(k^3+8k^2+11k)+\frac{k(k-1)}{2} = 72k^3+357.5k^2+263.5k+1$. ◀

Now we are ready to give our final reduction rule, the correctness of which follows from the above lemma.

▶ **Reduction Rule 9.** *Let (G', R', k') be the reduced instance after applying Reduction Rule 1-8 exhaustively. If $|V(G')| > 71k^3+349k^2+254k+2$, or $|E(G')| > 72k^3+357.5k^2+263.5k+1$, then return trivial* `No` *instance.*

It is easy to verify that every reduction rule can be executed within polynomial time. Therefore, we have a kernelization algorithm for Ext-IPOVD, that either correctly returns a trivial `Yes` instance or a trivial `No` instance, or produces an equivalent instance (G', R', k') of size $\mathcal{O}(k^3)$. On the reduced Ext-IPOVD instance, we apply Lemma 6 to construct an equivalent IPOVD instance $(H, k'+1)$ whose size is also bounded by $\mathcal{O}(k^3)$.

Now, we can conclude the kernelization algorithm with the following theorem.

▶ **Theorem 10.** IPOVD *admits a kernel of size $\mathcal{O}(k^3)$.*

5 Kernel Lower Bound for CPOVD

In this section, we show that CPOVD does not admit a polynomial kernel when parameterized by the solution size k, unless NP $\subseteq$ co-NP/poly. We establish this by providing a parameter-preserving reduction from the Connected Vertex Cover problem to CPOVD.

▶ **Theorem 11.** CPOVD *does not admit polynomial kernel unless* NP $\subseteq$ *co-*NP*/poly.*

Proof. We give a straightforward parameter-preserving reduction from Connected Vertex Cover to CPOVD. In Connected Vertex Cover, we are given an undirected graph G along with an integer k and our goal is to check whether there is a subset $X \subseteq V(G)$ of size at most k such that $G[S]$ is connected and X contains at least one end-point of each edge. It is kwown that Connected Vertex Cover does not admit a polynomial kernel unless NP $\subseteq$ co-NP/poly [9].

For an instance $(G = (V, E), k)$ of Connected Vertex Cover, we construct an instance $(G' = (V', E'), k')$ of CPOVD as follows: $V' = V(G) \cup \{z_{uv} : uv \in E(G)\}$, and $E' = E(G) \cup \{e^u_{uv} = uz_{uv}, e^v_{uv} = vz_{uv} : uv \in E(G)\}$, and we set $k' = k$. We claim that G has a connected vertex cover of size at most k if and only if G' has a connected pathwidth-one vertex deletion set of size at most k.

In the forward direction, let S is a solution to Connected Vertex Cover. Now consider the graph $G' - S$. As S is vertex cover, in $G' - S$, the degree of every vertex z_{uv} is at most one, because at least one of u or v belongs to S. Now as S is a vertex cover of G, the graph $G' - S$ is a disjoint union of stars. Clearly, $G'[S]$ is connected since $G[S]$ is connected. Hence S is a connected pathwidth-one vertex deletion set of size at most k in G'.

In the backward direction, let G' has a connected pathwidth-one vertex deletion set S' of size at most k. We show that the vertex set $S = S' \setminus \{z_{uv} \mid uv \in E(G)\}$ is a connected vertex cover of G of size at most k. By the construction of G', there is a cycle $C_{uv} = \{u, z_{uv}, v\}$ for every edge $uv \in E(G)$. Therefore, by Fact 1, every POVD solution must include at least one vertex from each C_{uv}. In other words, S' contains at least one vertex from C_{uv} for every edge $uv \in E(G)$. Suppose S' contains z_{uv} for some edge $uv \in E(G)$. Since $G'[S']$ is connected and z_{uv} is adjacent only to u and v in G', S' must also include at least one of u or v. On the other hand, if S' does not contain z_{uv} for some edge uv, then it must contain at least one of u or v to cover the cycle. Thus, in all cases, S' contains at least one endpoint of every edge in G, implying that $S = S' \setminus \{z_{uv} \mid uv \in E(G)\}$ is a vertex cover of G of size at most k. Finally, removing the z_{uv} vertices from S' does not disconnect the induced subgraph in G' or in G, since each z_{uv} is either a leaf or part of a cycle in $G'[S']$. Therefore, S is a connected vertex cover of G of size at most k. This completes the proof. ◀

6 FPT Algorithm for CPOVD

In this section, we obtain the following result for CPOVD.

▶ **Theorem 12.** CPOVD *can be solved in* $14^k \cdot n^{\mathcal{O}(1)}$ *time.*

Overview. By Fact 2, if a graph G does not contain T_2, K_3, or C_4 as a subgraph, then each connected component of G is either a tree or a cycle with zero or more hairs. To compute a set of vertices whose removal makes the graph free of T_2, K_3, and C_4 as subgraphs, we can simply identify such a structure in the graph by brute-force search (in n^7 time) and branch on it. This results in at most 7-way branching, leading to at most 7^k different possible paths from the root to the leaves in the branching tree. Also, notice that any solution to POVD as well as CPOVD must contain all the vertices of one such path as a subset. In this algorithm, we first branch on the forbidden structures (T_2, K_3, and C_4), and then we call the algorithm of Group Steiner Tree at most 7^k times (once for each branching path).

Before going into the details of the algorithm, let us see Group Steiner Tree problem.

Group Steiner Tree
Input: An undirected graph G, vertex-disjoint subsets $S_1, \ldots, S_\ell \subseteq V$, and $p \in \mathbb{N}$.
Question: Does G contain a tree on at most p vertices that intersects each S_i, $i \in [\ell]$?

The following result is known for the Group Steiner Tree problem.

▶ **Theorem 13.** [22, Lemma 3] Group Steiner Tree *can be solved in* $2^\ell \cdot n^{\mathcal{O}(1)}$ *time.*

Now we describe our algorithm for CPOVD in detail. For a given instance (G, k) of CPOVD, we first check whether G contains a T_2, K_3, or C_4 as a subgraph (by exhaustive search) and branch on such a structure if it exists. In each branching step, the parameter k is decreased by one. And update the graph to $G' = G - v$ and parameter to $k' = k - 1$. We stop branching on an instance (G', k') when either $k' \leq 0$ or G' contains none of T_2, K_3, or C_4 as a subgraph. Since T_2 contains 7 vertices, each branching step generate at most 7 sub-instances, leading to a 7-way branching. As k decreases by one at each step, the height of the branching tree is at most $k + 1$, and the total number of root-to-leaf paths is bounded by 7^k.

Let $\mathcal{P}$ denote the collection of vertex sets corresponding to each branching path, where each set $P_i \in \mathcal{P}$ consists of at most k vertices removed along that path. That is, if $P_i \in \mathcal{P}$, then there exists a root-to-leaf path in the branching tree such that P_i is the set of deleted vertices, and $|P_i| \leq k$. Clearly, $|\mathcal{P}| \leq 7^k$, and for each $P_i \in \mathcal{P}$, every connected component of $G - P_i$ is either a tree or a cycle with zero or more hairs. Now for each $P_i \in \mathcal{P}$, we check whether there is a connected povd solution of size at most k containing the vertex set P_i. Let $V(P_i) = \{v_{i_1}, v_{i_2}, \ldots, v_{i_{|P_i|}}\}$. Also Let $C^i_1, C^i_2, \ldots, C^i_{p_i}$ be the vertex sets corresponding to the cycle components in $G - P_i$. If $|P_i| + p_i > k$ then we immediately conclude that no such solution exists that contains the vertex set P_i. Else, i.e., when $|P_i| + p_i \leq k$, we invoke the Group Steiner Tree problem on the input graph G with the vertex-disjoint subsets $v_{i_1}, v_{i_2}, \ldots, v_{i_{|P_i|}}, C^i_1, C^i_2, \ldots, C^i_{p_i}$ with the budget parameter k. If Group Steiner Tree calls returns `Yes`, we return `Yes` for the input instance (G, k) of CPOVD; else if it returns `No`, then we immediately conclude that no such solution that contains the vertex set P_i. Now at the end, if for each $P_i \in \mathcal{P}$, we have that no such solution exists that contains the vertex set P_i we return `No` for the input instance (G, k) of CPOVD. We provide the correctness of the above-described algorithm for an input instance (G, k) of CPOVD in the Section A.2.

Proof of Theorem 12. We show that our algorithm for CPOVD runs in $\mathcal{O}^*(14^k)$ time. Recall that the branching step is at most a 7-way branching, producing $\mathcal{O}^*(7^k)$ nodes in the branching tree. At each node of the branching tree, we spend only polynomial time, since detecting a K_3, C_4, or T_2 subgraph, or reporting their absence, can be done in polynomial time. Therefore, the total time spent in branching is $\mathcal{O}^*(7^k)$. Since there are $\mathcal{O}^*(7^k)$ leaves in the branching tree, the number of distinct root-to-leaf paths is also bounded by $\mathcal{O}^*(7^k)$, and thus $|\mathcal{P}| = \mathcal{O}^*(7^k)$. Consequently, we make at most $\mathcal{O}^*(7^k)$ calls to Group Steiner Tree. For any $P_i \in \mathcal{P}$, we call Group Steiner Tree only if $|P_i| + p \leq k$. By Theorem 13, each such Group Steiner Tree instance can be solved in $\mathcal{O}^*(2^{|P_i|+p})$, that is, in $\mathcal{O}^*(2^k)$ time. Therefore, solving all Group Steiner Tree instances requires at most $\mathcal{O}^*(7^k \cdot 2^k) = \mathcal{O}^*(14^k)$ time. Combining the branching and the Group Steiner Tree solving phases, the total running time is bounded by $\mathcal{O}^*(7^k) + \mathcal{O}^*(14^k) = \mathcal{O}^*(14^k)$. Hence, the proof. ◀

7 FPT Algorithm for IPOVD

In this section, we present an FPT algorithm for IPOVD with a running time of $7^k \cdot k^{\mathcal{O}(1)} + n^{\mathcal{O}(1)}$. This algorithm also begins by branching on occurrences of T_2, K_3, and C_4 subgraphs. Once this branching process finishes, we have at most 7^k root-to-leaf paths in the branching tree, and every minimal independent povd solution must fully include one of these paths. For each such path P_i, we check whether it can be extended to a solution to IPOVD, where a povd S is said to be a solution to IPOVD if $|S| \leq k$ and S is an independent set in G. This verification for each fixed P_i can be done in polynomial time.

Algorithm. We now describe the algorithm in more detail. Given an instance (G, k) of IPOVD, we iteratively remove occurrences of T_2, K_3, and C_4 by branching. Whenever such a structure exists in G, we branch over its vertices, reducing k by one at each branching step. Branching stops when we reach to an instance (G', k') where either $k' \leq 0$ or G' no longer contains any of these subgraphs. Once the branching tree is constructed, we process each root-to-leaf path P_i as follows: we check whether P_i forms an independent set, and whether every cycle in $G - P_i$ has at least one vertex with no neighbors in P_i. If this is true, we select exactly one such vertex from each cycle and collect them into a set S_i. If $|S_i \cup V(P_i)| \leq k$, we return $S_i \cup V(P_i)$ as a valid IPOVD solution. If no such path extends to such a solution, we return that (G, k) is a `No` instance of IPOVD.

Correctness. As established earlier, there are at most 7^k different root-to-leaf paths in the branching tree. Let $\mathcal{P}$ be the collection of these paths. Every minimal IPOVD solution must completely contain one of the paths in $\mathcal{P}$. Therefore, for each $P_i \in \mathcal{P}$, we test whether it can be extended to a solution of size at most k. Observe that this requires P_i to be an independent set. Since $G - P_i$ contains no T_2, K_3, or C_4, by Fact 2 each connected component of $G - P_i$ is either a tree or a cycle (possibly with pendant vertices). Let c_i be the number of cycles in $G - P_i$. Any solution must pick at least one vertex from each such cycle. Therefore, constructing S_i by selecting one vertex per cycle with no neighbors in P_i suffices. Since the cycles are disjoint, we do not need to worry about independence among vertices in S_i. This establishes the correctness of the algorithm.

Running Time. Constructing the branching tree takes time $\mathcal{O}^*(7^k)$. For each of the 7^k paths P_i, checking whether it extends to a valid solution requires only polynomial time. Thus, the total time is $\mathcal{O}^*(7^k)$. Additionally, since a polynomial kernel of size $\mathcal{O}(k^3)$ exists for IPOVD, we first run kernelization on the input instance (G, k). If kernelization returns a trivial `Yes` or `No` instance, we return the corresponding answer `Yes` or `No`, respectively, to IPOVD. Otherwise, it produces an equivalent instance (G', k') with $\mathcal{O}(k^3)$ vertices, to which we apply the algorithm above. This results in a total running time of $7^k \cdot k^{\mathcal{O}(1)} + n^{\mathcal{O}(1)}$. Hence, we obtain the following theorem.

▶ **Theorem 14.** IPOVD *admits an* FPT *algorithm with running time* $7^k \cdot k^{\mathcal{O}(1)} + n^{\mathcal{O}(1)}$*.*

8 Conclusion

We studied two variants of the POVD problem: the connected version (CPOVD) and the independent version (IPOVD). We showed that CPOVD is solvable in $\mathcal{O}^*(14^k)$ time but admits no polynomial kernel unless NP $\subseteq$ co-NP/poly. In contrast, IPOVD can be solved in $\mathcal{O}^*(7^k)$ time with a polynomial kernel of size $\mathcal{O}(k^3)$. Improving these running times, reducing the kernel size for IPOVD, improving the polynomial-time approximation factor, or establishing lower bounds are interesting avenues for future research.

References

1 Ankit Abhinav, Satyabrata Jana, Nidhi Purohit, Abhishek Sahu, and Saket Saurabh. Parameterized complexity of feedback vertex set with connectivity constraints. In Rastislav Královic and Vera Kurková, editors, *SOFSEM 2025: Theory and Practice of Computer Science - 50th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2025, Bratislava, Slovak Republic, January 20-23, 2025, Proceedings, Part I*, volume 15538 of *Lecture Notes in Computer Science*, pages 23–36. Springer, 2025. `doi:10.1007/978-3-031-82670-2_3`.

2 Akanksha Agrawal, Sushmita Gupta, Saket Saurabh, and Roohani Sharma. Improved algorithms and combinatorial bounds for independent feedback vertex set. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 2:1–2:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.IPEC.2016.2`.

3 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discret. Math.*, 12(3):289–297, 1999. `doi:10.1137/S0895480196305124`.

4 Fabián A. Chudak, Michel X. Goemans, Dorit S. Hochbaum, and David P. Williamson. A primal-dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Oper. Res. Lett.*, 22(4-5):111–118, 1998. `doi:10.1016/S0167-6377(98)00021-2`.

5 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015. `doi:10.1007/978-3-319-21275-3`.

6 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011. `doi:10.1109/FOCS.2011.23`.

7 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. `doi:10.1145/3506707`.

8 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. An improved FPT algorithm and a quadratic kernel for pathwidth one vertex deletion. *Algorithmica*, 64(1):170–188, 2012. `doi:10.1007/s00453-011-9578-2`.

9 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and ids. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2009. `doi:10.1007/978-3-642-02927-1_32`.

10 Baris Can Esmer and Ariel Kulik. Sampling with a black box: Faster parameterized approximation algorithms for vertex deletion problems. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, July 8-11, 2025, Aarhus, Denmark*, volume 334 of *LIPIcs*, pages 39:1–39:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. `doi:10.4230/LIPICS.ICALP.2025.39`.

11 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Solving connected dominating set faster than 2^{n}. *Algorithmica*, 52(2):153–166, 2008. `doi:10.1007/S00453-007-9145-Z`.

12 Yoichi Iwata and Yusuke Kobayashi. Improved analysis of highest-degree branching for feedback vertex set. *Algorithmica*, 83(8):2503–2520, 2021. `doi:10.1007/s00453-021-00815-w`.

13 Satyabrata Jana, Daniel Lokshtanov, Soumen Mandal, Ashutosh Rai, and Saket Saurabh. Parameterized approximation scheme for feedback vertex set. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 56:1–56:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.MFCS.2023.56`.

14 R. Krithika, Diptapriyo Majumdar, and Venkatesh Raman. Revisiting connected vertex cover: FPT algorithms and lossy kernels. *Theory Comput. Syst.*, 62(8):1690–1714, 2018. `doi:10.1007/S00224-017-9837-Y`.

15 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`.

16 Jason Li and Jesper Nederlof. Detecting feedback vertex sets of size k in $O^\star(2.7k)$ time. *ACM Trans. Algorithms*, 18(4):34:1–34:26, 2022. `doi:10.1145/3504027`.

17 Shaohua Li and Marcin Pilipczuk. An improved FPT algorithm for independent feedback vertex set. *Theory Comput. Syst.*, 64(8):1317–1330, 2020. doi:10.1007/s00224-020-09973-w.

18 Ching-Hao Liu, Sheung-Hung Poon, and Jin-Yong Lin. Independent dominating set problem revisited. *Theor. Comput. Sci.*, 562:1–22, 2015. doi:10.1016/J.TCS.2014.09.001.

19 Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Covering small independent sets and separators with applications to parameterized algorithms. *ACM Trans. Algorithms*, 16(3):32:1–32:31, 2020. doi:10.1145/3379698.

20 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Trans. Algorithms*, 9(4):30:1–30:35, 2013. doi:10.1145/2500119.

21 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theor. Comput. Sci.*, 461:65–75, 2012. doi:10.1016/j.tcs.2012.02.012.

22 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT algorithms for connected feedback vertex set. *J. Comb. Optim.*, 24(2):131–146, 2012. doi:10.1007/s10878-011-9394-2.

23 Daniel Mölle, Stefan Richter, and Peter Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. *Theory Comput. Syst.*, 43(2):234–253, 2008. doi:10.1007/S00224-007-9089-3.

24 Geevarghese Philip, Venkatesh Raman, and Yngve Villanger. A quartic kernel for pathwidth-one vertex deletion. In Dimitrios M. Thilikos, editor, *Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, Zarós, Crete, Greece, June 28-30, 2010 Revised Papers*, volume 6410 of *Lecture Notes in Computer Science*, pages 196–207, 2010. doi:10.1007/978-3-642-16926-7_19.

25 Johannes Rauch, Dieter Rautenbach, and Uéverton S. Souza. Exact and parameterized algorithms for the independent cutset problem. *J. Comput. Syst. Sci.*, 148:103598, 2025. doi:10.1016/J.JCSS.2024.103598.

26 Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *J. Comb. Theory B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

27 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

28 Yinglei Song. An improved parameterized algorithm for the independent feedback vertex set problem. *Theor. Comput. Sci.*, 535:25–30, 2014. doi:10.1016/j.tcs.2014.03.031.

29 Dekel Tsur. Faster algorithm for pathwidth one vertex deletion. *Theor. Comput. Sci.*, 921:63–74, 2022. doi:10.1016/j.tcs.2022.04.001.

A Appendix

A.1 Correctness of Reduction Rules 4-8

▶ **Lemma 15.** *Reduction Rule 4 is correct.*

Proof. Let P_v denote the set of all pendant neighbors of v in G, and let $D_v \subseteq P_v$ be the set of pendant neighbors of v that are deleted from G to obtain G'. Since G' is a subgraph of G and $R' \subseteq R$, if (G, R, k) is a `Yes` instance, then (G', R', k) is also a `Yes` instance. For the reverse direction, assume that (G', R', k) is a `Yes` instance. Then there exists a solution S for G' such that $|S| \leq k$ and $S \cap R' = \emptyset$. We claim that S is also a solution for G and satisfies $S \cap R = \emptyset$.

Suppose, for the sake of contradiction, that S is not a solution for G. Then there exists a structure T_2 (say, Q) in G such that $S \cap Q = \emptyset$. Since S is a solution for G', it follows that Q must contain some vertex $u \in Q \cap D_v$. Because $|S| \leq k$ and $|P_v \setminus D_v| = k + 1$, there exists a vertex $u' \in P_v \setminus D_v$ such that $u' \notin S$. Let $Q' = (Q \setminus \{u\}) \cup \{u'\}$. Then Q' is also a T_2 in G' satisfying $Q' \cap S = \emptyset$, which contradicts the assumption that S is a solution for G'.

Finally, since $S \cap D_v = \emptyset$, $R = R' \cup D_v$, and $S \cap R' = \emptyset$, we have $S \cap R = \emptyset$. Therefore, S is a solution for G of size at most k and disjoint from R, implying that (G, R, k) is a `Yes` instance. This completes the proof. ◀

▶ **Lemma 16.** *Reduction Rule 5 is correct.*

Proof. Notice that, in this case, if there is a solution of size at most k, then that solution must include u. Otherwise, to eliminate all the T_2 in $G[V(M) \cup \{u\}]$, we would need to delete strictly more than k vertices. Thus, if $u \in R$, there cannot be a k size solution disjoint from R. On the other hand, if $u \notin R$, then deleting u, decreasing k by one, and adding $N(u)$ to R is safe, since u necessarily belongs to any valid solution. ◀

▶ **Lemma 17.** *Reduction Rule 6 is correct.*

Proof. First, we assume that (G, R, k) is a `Yes` instance. We want to show that (G', R', k) is also a `Yes` instance. Let S be an ipovd of size at most k disjoint from R. There are two cases to consider: either $S \cap \{v_3, v_4\} = \emptyset$ or exactly one of v_3 and v_4 belongs to S. If $S \cap \{v_3, v_4\} = \emptyset$, then the same set S is an ipovd for G' of size at most k disjoint from R, since the contraction does not create any new cycles or new T_2. Next, suppose exactly one of v_3 or v_4 belongs to S. Then the contracted vertex $v_3^4 \notin R'$ in G'. Notice that v_3 and v_4 cannot be part of any T_2, and if they belong to any cycle, then v_2, v_3, v_4, and v_5 all participate in that cycle. Therefore, if either v_2 or v_5 is in S, then the set $S \setminus \{v_3, v_4\}$ is an ipovd for G' of size at most k disjoint from R. Otherwise, the set $S' = \big(S \setminus \{v_3, v_4\}\big) \cup \{v_3^4\}$ is an ipovd for G' of size at most k disjoint from R'.

In converse direction, assume that (G', R', k) is a `Yes` instance, and let S' be a solution for (G', R', k) of size at most k. If $v_3^4 \notin S'$, then by Observation 7, the set $S = S'$ is a solution for (G, R, k) of size at most k. Otherwise, if $v_3^4 \in S'$, then at least one of v_3 or v_4 was not in R in G. Let that vertex be v_i, where i is either 3 or 4. Then again by Observation 7, the set $S = \big(S' \setminus \{v_3^4\}\big) \cup \{v_i\}$ is a solution for (G, R, k) of size at most k. ◀

▶ **Lemma 18.** *Reduction Rule 7 is correct.*

Proof. The correctness of Reduction Rule 7 is quite similar to the correctness of Reduction Rule 5, since any POVD solution must cover all cycles, and thus any k size POVD solution must include u in this case. Therefore, if $u \in R$, there cannot be a k size POVD solution. Otherwise, if $u \notin R$, then deleting u, decreasing k by one, and adding $N(u)$ to R is safe, as u necessarily belongs to any valid solution. ◀

▶ **Lemma 19.** *Reduction Rule 8 is correct.*

Proof. First, we will show that if S is a minimal ipovd of size k for G disjoint from R, then S is also a ipovd of size k for G' and remains disjoint from R'. To establish this, it suffices to show that $S \cap Y' = \emptyset$.

For contradiction, suppose that $y_i^x \in S$ for some $y_i^x \in Y'$. Since S is an independent set, both u and x are not in S. Also observe that in $G\big[V \setminus \big(S \setminus \phi(x)\big)\big]$, there are $k+2$ vertex-disjoint paths between u and x, each passing through a different vertex of $\phi(x)$. This implies that at least $k+1$ vertices of $\phi(x)$ must be included in S, which contradicts the assumption that $|S| \le k$. Therefore, we have $S \cap Y' = \emptyset$.

For the converse direction, let S' be an ipovd of size k for the graph G', disjoint from R'. We need to show that S' is an ipovd for G as well. Note that by the way Reduction Rule 8 has been applied, for each $x \in X'$, there exist indices i and j such that the cycle (u, y_i^x, x, y_j^x, u) is present in G'. To hit this cycle, at least one of the vertices u or x must belong to S'. Since $S' \cap Y' = \emptyset$, the set S' is independent and remains disjoint from R.

Now, suppose for contradiction that S' is not a ipovd in G. Then there must be some cycle or T_2 in $G[V \setminus S']$. Since there was no such cycle or T_2 in $G'[V \setminus S']$, any such structure in G must include an edge of the form uy_i^x for some i. However, for each x, there are exactly two such edges in G', and their presence would imply a cycle or T_2 in G', contradicting our assumption. Hence, S' is indeed a ipovd of size k for G disjoint from R. ◀

A.2 Missing proof details for Theorem 12

Correctness. We will argue the correctness of the above-described algorithm for an input instance (G, k) of CPOVD in the following two claims.

▷ Claim 20. The algorithm returns `Yes` if and only if (G, k) is a `Yes` instance.

Proof. If the algorithm returns `Yes`, then at least one call to Group Steiner Tree must have returned `Yes`, corresponding to some $P_j \in \mathcal{P}$. If S is a solution to the Group Steiner Tree instance for this call, then by the way the instance was constructed, we have $P_j \subseteq S$, and S also contains at least one vertex from each cycle in $G - P_j$. Thus, $G - S$ is acyclic and T_2-free (since $G - P_j$ is K_3, C_4, and T_2-free). Moreover, by the properties of the Group Steiner Tree solution corresponding to P_j, $G[S]$ is connected and $|S| \leq k$. Hence, S is a valid solution for CPOVD.

If the algorithm returns `No` because for each $P_i \in \mathcal{P}$ either $|P_i| + p_i > k$ or the algorithm return `No` when it call to corresponding Group Steiner Tree. In the first case, if $|P_i| + p_i > k$, then there cannot be a size-k pathwidth-one vertex deletion set, since any such set must include all vertices of some P_i together with at least one vertex from each of the p cycles in $G - P_i$, exceeding the budget k. If second case, the algorithm must made a call to Group Steiner Tree and returned `No`, Towards contradiction, suppose there exists a solution S to CPOVD for (G, k). Then there must be some $P_j \in \mathcal{P}$ such that $P_j \subseteq S$. Moreover, S must contain at least one vertex from each cycle in $G - P_j$. Since S is a solution to CPOVD, we have that $G[S]$ is connected and $|S| \leq k$. This implies that S would be a solution to the Group Steiner Tree instance corresponding to P_j, contradicting the assumption that every call to Group Steiner Tree returned `No`. ◁