Parameterized Complexity of Deletion to Scattered Graph Classes

Ashwin Jacob
The Institute of Mathematical Sciences, HBNI, Chennai, India
ajacob@imsc.res.in

Diptapriyo Majumdar
Royal Holloway, University of London, UK
diptapriyo.majumdar@rhul.ac.uk

Venkatesh Raman
The Institute of Mathematical Sciences, HBNI, Chennai, India
vraman@imsc.res.in

Abstract

Graph-modification problems, where we add/delete a small number of vertices/edges to make the given graph to belong to a simpler graph class, is a well-studied optimization problem in all algorithmic paradigms including classical, approximation and parameterized complexity. Specifically, graph-deletion problems, where one needs to delete at most k vertices to place it in a given non-trivial hereditary (closed under induced subgraphs) graph class, captures several well-studied problems including Vertex Cover, Feedback Vertex Set, Odd Cycle Transveral, Cluster Vertex Deletion, and Perfect Deletion. Investigation into these problems in parameterized complexity has given rise to powerful tools and techniques. While a precise characterization of the graph classes for which the problem is *fixed-parameter tractable* (FPT) is elusive, it has long been known that if the graph class is characterized by a *finite* set of forbidden graphs, then the problem is FPT.

In this paper, we initiate a study of a natural variation of the problem of deletion to *scattered graph classes* where we need to delete at most k vertices so that in the resulting graph, each connected component belongs to one of a constant number of graph classes. A simple hitting set based approach is no longer feasible even if each of the graph classes is characterized by finite forbidden sets. As our main result, we show that this problem (in the case where each graph class has a finite forbidden set) is fixed-parameter tractable by a $O^*(2^{k^{\mathcal{O}(1)}})$[1] algorithm, using a combination of the well-known techniques in parameterized complexity – *iterative compression* and *important separators*. Our approach follows closely that of a related problem in the context of satisfiability [Ganian, Ramanujan, Szeider, TAlg 2017], where one wants to find a small backdoor set so that the resulting CSP (constraint satisfaction problem) instance belongs to one of several *easy* instances of satisfiability. While we follow the main idea from this work, there are some challenges for our problem which we needed to overcome.

When there are two graph classes with finite forbidden sets to get to, and if one of the forbidden sets has a path, then we show that the problem has a (better) singly exponential algorithm and a polynomial sized kernel. We also design an efficient FPT algorithm for a special case when one of the graph classes has an infinite forbidden set. Specifically, we give a $O^*(4^k)$ algorithm to determine whether k vertices can be deleted from a given graph so that in the resulting graph, each connected component is a tree (the sparsest connected graph) or a clique (the densest connected graph).

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Parameterized Complexity, Scattered Graph Classes, Important Separators

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.18

Acknowledgements The authors thank M.S.Ramanujan for many helpful discussions.

[1] The O^* notation ignores polynomial factors.

1 Introduction

Graph modification problems, where we want to modify a given graph by adding/deleting vertices/edges to obtain a *simpler* graph are well-studied problems in algorithmic graph theory. Starting from the classical work of Lewis and Yannakakis [12] (see also [20]) who showed the problem NP-COMPLETE for the resulting simpler graph belonging to any non-trivial (the graph property is true for infinitely many graphs and false for infinitely many graphs) hereditary graph class (closed under induced subgraphs), the complexity of the problem has been studied in several algorithmic paradigms including approximation and parameterized complexity. Specifically, deleting at most k vertices to a fixed hereditary graph class is an active area of research in parameterized complexity over the last several years yielding several powerful tools and techniques. Examples of such problems include VERTEX COVER, CLUSTER VERTEX DELETION, FEEDBACK VERTEX SET and CHORDAL VERTEX DELETION.

It is well-known that any hereditary graph class can be described by a forbidden set of graphs, finite or infinite, that contains all minimal forbidden graphs in the class. In parameterized complexity, it is known that the deletion problem is fixed-parameter tractable (FPT) as long as the resulting hereditary graph class has a finite forbidden set [3]. This is shown by an easy reduction to the BOUNDED HITTING SET problem. This includes, for example, deletion to obtain a split graph or a cograph. We also know FPT algorithms for specific graph classes defined by infinite forbidden sets like feedback vertex set and odd cycle transversal [6]. While the precise characterization of the class of graphs for which the deletion problem is FPT is elusive, there are graph classes for which the problem is W-hard [10, 13].

Recently, some stronger versions have also been studied, where the problem is to delete at most k vertices to get a graph such that every connected component of the resulting graph is at most ℓ edges away from being a graph in a graph class $\mathcal{F}$ (see [16–18]). Some examples of $\mathcal{F}$ that have been studied in this stronger version include *forest, pseudo-forest or bipartite.*

Our results: In this paper, we address the complexity of a very natural variation of the graph deletion problem, where in the resulting graph, each connected component belongs to one of finitely many graph classes. For example, we may want the connected components of the resulting graph to be a clique or a biclique (a complete bipartite graph). It is known that cliques forbid exactly P_3s, the induced paths of length 2, and bicliques forbid P_4 and triangles. So if we just want every connected component to be a clique or every connected component to be a biclique, then one can find appropriate constant sized subgraphs in the given graph and branch on them (as one would in a hitting set instance). However, if we want each connected component to be a clique or a biclique, such a simple approach by branching over P_3, P_4, or K_3 would not work. Notice that triangles are allowed to be present in clique components and P_3s are allowed to be present in biclique components. It is not even clear that there will be a finite forbidden set for this resulting graph class.

Our main result of the paper shows this deletion problem, when there are a constant number of graph classes each characterized by a finite forbidden set for the resulting graph is fixed-parameter tractable using the well-known techniques in parameterized complexity – iterative compression and important separators. Specifically the problem we show to be fixed-parameter tractable is the following.

$(\Pi_1, \Pi_2, \ldots, \Pi_d)$ VERTEX DELETION **Parameter:** k
Input: An undirected graph $G = (V, E)$, an integer k, and d graph classes $\Pi_1, \ldots, \Pi_d$ described by finite forbidden sets $\mathcal{F}_1, \ldots, \mathcal{F}_d$ respectively.
Question: Is there a subset $Z \subseteq V(G)$, $|Z| \leq k$ such that every connected component of $G - Z$ is in at least one of the graph classes $\Pi_1, \ldots, \Pi_d$?

Several natural graph classes forbid induced paths of certain lengths. We can develop a much better algorithm in that case. More specifically, when there are only two graph classes for the connected components of the resulting graph to be placed, and if one of them has a path as a forbidden subgraph, then we show that the problem has a polynomial kernel, an efficient $O^*(c^k)$ algorithm and a c-approximation algorithm where c depends on the maximum size of the graphs in the forbidden set. Finally we also look at the problem for the specific case when one of the resulting graph classes has an infinite forbidden set. Specifically, we look at the problem of deleting a small number of vertices so that each connected component of the resulting graph is a tree (the sparsest connected graph) or a clique (the densest connected graph), and give an $O^*(4^k)$ algorithm.

Previous Work. While there has been a lot of work on graph deletion and modification problems, one work that comes close to ours is the work by Ganian, Ramanujan and Szeider [9] where they consider the parameterized complexity of finding strong backdoors to a scattered class of CSP instances. In fact, in their conclusion, they remark that

> "graph modification problems and in particular the study of efficiently computable modulators to various graph classes has been an integral part of parameterized complexity and has led to the development of several powerful tools and techniques. We believe that the study of modulators to "scattered graph classes" could prove equally fruitful and, as our techniques are mostly graph based, our results as well as techniques could provide a useful starting point towards future research in this direction".

Our work is a starting point in addressing the parameterized complexity of the problem they suggest.

Our Techniques. We now give a brief summary of the main FPT algorithm described in Theorem 3.3. We reduce the problem $(\Pi_1, \Pi_2, \ldots, \Pi_d)$ Vertex Deletion to Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$ Vertex Deletion Compression (Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC for short) using the standard technique of iterative compression. In Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC, we can assume that a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator W of the graph of size $k - i$ is also given to us as input for some $i \leq k$ (i is the number of vertices from the modulator we have guessed to be in the solution) and the aim is to check if there is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator of the graph of size $k - i - 1$ disjoint from W. This is formally described in Subsection 3.1.

In Subsection 3.2, we give an FPT algorithm for Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC in the special case when the solution that we are looking for leaves W in a single component. The algorithm uses the technique of important separators [15].

Finally in Subsection 3.3, we handle general instances of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. We focus on instances where the solution separates W. We guess $W_1 \subset W$ as the part of W that occurs in some single connected component after deleting the solution. Since, the solution separates W, we know that it contains a $W_1 - (W \setminus W_1)$ separator X. The algorithm uses the technique of important separator sequences [14]. Informally, an important separator sequence partitions the graph into slices with small boundaries which allows us to look for solutions local to the slices. The algorithm guesses the integer ℓ which is the size of the part of the solution present in the graph containing W_1 after removing X. The algorithm then constructs the important separator sequence corresponding to ℓ and finds the separator P furthest from W_1 in the sequence such that there is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator of size ℓ in the graph containing W_1 after removing P. If separator X either intersects P or dominates the other, then a recursive smaller instance is easily constructable. In the case when the two separators are incomparable, the algorithm identifies a set of vertices Y that is reachable from W_1 after deleting P. The algorithm then constructs a graph gadget of $k^{\mathcal{O}(1)}$ vertices whose appropriate attachment to the boundary P of the graph $G[Y]$ gives a graph G' which preserves the part of the solution of G present in $G[Y]$. Since this part of the solution is strictly smaller in size, the algorithm can find the solution for G by recursively finding the solution in G'.

While the main techniques for a good part follow from the paper by Ganian, Ramanujan and Szeider [9], the problem has its own challenges. The paper by Ganian, Ramanujan and Szeider worked on strong backdoors to a scattered class of CSPs. The authors create an variable-constraint incidence graph based on the CSP and focus on “forbidden sets” with respect to a set of variables S which is a set of constraints C such that there is an assignment of S on which for every CSP class, one of the constraints under this assignment does not belong to this class. We identify an equivalent notion of forbidden set in our problem as a subset of vertices C such that for every finite forbidden graph class, there is a subset of C such that the graph induced on the subset belongs to a forbidden member of this class. Since the forbidden set is identified by a collection of finite subgraphs here, instead of a set of constraint vertices in the CSP case where the graph properties are not relevant, there are more difficulties arising from it. For example, the CSP result uses a connecting gadget that adds extra vertices and edges to preserve connectivity of some subsets. They manage this by associating tautological constraints to these extra vertices which can go into any of the CSP classes. But adding new vertices and edges can create new subgraphs which could be a finite forbidden subgraph, in our case. Due to this, we had to come up with an algorithm avoiding any use of connecting gadgets. Another difficulty is in creating the smaller graph instance G' using gadgets as described in the previous paragraph where we could recursively solve the instance. In the CSP case, identifying the set of vertices that need to be preserved in the smaller instance was easier as the forbidden sets were a set of constraint vertices. In our case, this gets more complicated as we deal with forbidden sets involving subgraphs. Specifically this results in a more complex marking procedure in Lemma 3.17 to prove its third claim.

2 Preliminaries

Graph Theory. For $\ell \in \mathbb{N}$, we use P_ℓ to denote the path on ℓ vertices. We use standard graph theoretic terminology from Diestel’s book [8]. A *tree* is a connected graph with no cycles. A *forest* is a graph, every connected component of which is a tree. A *paw* is a graph G with vertex set $V(G) = \{x_1, x_2, x_3, x_4\}$ and edge set $E(G) = \{x_1x_2, x_2x_3, x_3x_1, x_3x_4\}$. A graph is a *block graph* if all its biconnected components are cliques. For a set $X \subseteq G$, we use $G[X]$ to denote the graph induced on the vertex set X and we use $G - X$ (or $G \setminus X$) to denote the graph induced by the vertex set $V(G) \setminus X$. We say that a subset $Z \subseteq V(G)$ *disconnects* a subset $S \subseteq V(G)$ if there exists $v, w \in S$ with $v \neq w$ such that v and w occur in different connected components of the graph $G \setminus Z$.

▶ **Definition 2.1.** *Let G be a graph and disjoint subsets $X, S \subseteq V(G)$. We denote by $R_G(X, S)$ the set of vertices that lie in the connected component containing X in the graph $G \setminus S$. We denote $R_G[X, S] = R_G(X, S) \cup S$. Finally we denote $NR_G(X, S) = V(G) \setminus R_G[X, S]$ and $NR_G[X, S] = NR_G(X, S) \cup S$. We drop the subscript G if it is clear from the context.*

▶ **Definition 2.2** ([15]). *Let G be a graph and $X, Y \subseteq V(G)$.*

- *A vertex set S disjoint from X and Y is said to disconnect X and Y if $R_G(X, S) \cap Y = \phi$. We say that S is an $X - Y$ **separator** in the graph G.*

- *An $X - Y$ separator is **minimal** if none of its proper subsets is an $X - Y$ separator.*
- *An $X - Y$ separator S_1 is said to **cover** an $X - Y$ separator S with respect to X if $R(X, S) \subset R(X, S_1)$.*
- *Two $X - Y$ separators S_1 and S_2 are said to be incomparable if neither covers the other.*
- *In a set $\mathcal{H}$ of $X - Y$ separators, a separator S is said to be component-maximal if there is no separator S' in $\mathcal{H}$ which covers S. Component-minimality is defined analogously.*
- *An $X - Y$ separator S_1 is said to dominate an $X - Y$ separator S with respect to X if $|S_1| \leq |S|$ and S_1 covers S with respect to X.*
- *We say that S is an important $X - Y$ separator if it is minimal and there is no $X - Y$ separator dominating S with respect to X.*

For the basic definitions of Parameterized Complexity, we refer to [6].

3 Deletion to d finite scattered classes

▶ **Definition 3.1.** *We call a set Z a* **$(\Pi_1, \Pi_2, \dots, \Pi_d)$-modulator** *if every connected component of $G \setminus Z$ is in one of the graph classes Π_i for $i \in d$.*

Organization of the section. This section is divided into three subsections. In the first section, we use iterative compression to transform the $(\Pi_1, \Pi_2, \dots, \Pi_d)$ Vertex Deletion problem into a compressed version of the problem named Disjoint $(\Pi_1, \Pi_2, \dots, \Pi_d)$-VDC. In the second section, we give an algorithm for Disjoint $(\Pi_1, \Pi_2, \dots, \Pi_d)$-VDC for the special case of having a non-separating solution which we will define later. Finally in the third section, we give a general algorithm for Disjoint $(\Pi_1, \Pi_2, \dots, \Pi_d)$-VDC and subsequently $(\Pi_1, \Pi_2, \dots, \Pi_d)$ Vertex Deletion using the algorithm in the second section as a subroutine.

3.1 Iterative Compression

We use the standard technique of iterative compression to transform the $(\Pi_1, \Pi_2, \dots, \Pi_d)$ Vertex Deletion problem into the following problem Disjoint $(\Pi_1, \Pi_2, \dots, \Pi_d)$ Vertex Deletion Compression(Disjoint $(\Pi_1, \Pi_2, \dots, \Pi_d)$-VDC) such that an FPT algorithm with running time $\mathcal{O}^*(f(k))$ for the latter gives a $\mathcal{O}^*(2^{k+1} f(k))$ time algorithm for the former. The details are moved to the full version.

Disjoint $(\Pi_1, \Pi_2, \dots, \Pi_d)$-VDC **Parameter:** k
Input: A graph G, an integer k, finite forbidden sets $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_d$ for graph classes $\Pi_1, \Pi_2, \dots, \Pi_d$ and a subset W of $V(G)$ such that W is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$- modulator of size $k + 1$.
Question: Is there a subset $Z \subseteq V(G) \setminus W$, $|Z| \leq k$ such that Z is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$-modulator of the graph G?

3.2 Finding non-separating solutions

In this section, we focus on solving instances of Disjoint $(\Pi_1, \Pi_2, \dots, \Pi_d)$-VDC which have a non-separating property defined as follows.

▶ **Definition 3.2.** *Let (G, k, W) be an instance of* Disjoint $(\Pi_1, \Pi_2, \dots, \Pi_d)$-VDC *and Z be a solution for this instance. Then Z is called a **non-separating** solution if W is contained in a single connected component of $G \setminus Z$ and **separating** otherwise. If an instance has only separating solutions, we call it a separating instance. Otherwise, we call it non-separating.*

We begin the description of the algorithm with the following reduction rule.

▶ **Reduction Rule 1.** *If a connected component of G belongs to some graph class Π_i, then remove all the vertices of this connected component.*

▶ **Lemma 3.3** ($\star$[2])**.** *Reduction rule 1 is safe.*

We now develop the following notion of forbidden sets which can be used to identify if a connected component of a graph belongs to any of the classes Π_i for $i \in [d]$.

▶ **Definition 3.4.** *We say that a subset of vertices $C \subseteq V(G)$ is a* ***forbidden set*** *of G if C occurs in a connected component of G and there exists a subset $C_i \subseteq C$ such that $G[C_i] \in \mathcal{F}_i$ for all $i \in [d]$ and C is a minimal such set.*

Clearly if a connected component of G contains a forbidden set, then it does not belong to any of the graph classes Π_i for $i \in [d]$. We note that even though the forbidden set C is of finite size, the lemma below rules out the possibility of a simple algorithm involving just branching over all the vertices of C.

▶ **Lemma 3.5** ($\star$)**.** *Let G be a graph and $C \subseteq V(G)$ be a forbidden set of G. Let Z be a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator of G. Then Z disconnects C or $Z \cap C \neq \emptyset$.*

We now have the following lemma on important separators which is helpful in our algorithm to compute non-separating solutions.

▶ **Lemma 3.6** ([4])**.** *For every $k \geq 0$ and subsets $X, Y \subseteq V(G)$, there are at most 4^k important $X - Y$ separators of size at most k. Furthermore, there is an algorithm that runs in $\mathcal{O}(4^k kn)$ time that enumerates all such important $X - Y$ separators and there is an algorithm that runs in $n^{\mathcal{O}(1)}$ time that outputs one arbitrary component-maximal $X - Y$ separator.*

We now have the following lemma which connects the notion of important separators with non-separating solutions of our problem.

▶ **Lemma 3.7** ($\star$)**.** *Let (G, k, W) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *obtained after exhaustively applying Reduction Rule 1 and Z be a non-separating solution. Let v be a vertex such that Z is a $\{v\} - W$ separator. Then there is a solution Z' which contains an important $v - W$ separator of size at most k in G.*

We use the above lemma along with Lemma 3.6 to obtain our algorithm for non-separating instances. The algorithm finds a minimal forbidden set C in polynomial time which is finite. Then it branches on the set C and also on $v - W$ important separators of size at most k of G for all $v \in C$.

▶ **Lemma 3.8.** *Let (G, k, W) be a non-separating instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. *Then it can be solved in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time where $|V| = n$.*

Proof. We first apply Reduction Rule 1 exhaustively. If the graph is empty, we return YES. Else, there is a connected component of G which does not belong to any graph classes Π_i for $i \in [d]$. Therefore, there exists a forbidden set $C \subseteq V(G)$ of G present in this connected component. We find C by checking for each graph class Π_i, if a graph in $\mathcal{F}_i$ exists as an induced subgraph in a any connected component $\mathcal{X}$ of G, take the union of these vertices and make it minimal by removing vertices and seeing if the set still remains forbidden. We

[2] The proofs of theorems and lemmas marked ($\star$) are moved to the full version due to lack of space.

branch in $|C|$-many ways by going over all the vertices $v \in C$ and in each branch, recurse on the instance $(G - v, k - 1, W)$. Then for all $v \in C$, we branch over all important $v - W$ separators X of size at most k in G and recurse on instances $(G \setminus X, k - |X|, W)$.

We now prove the correctness of the algorithm. Let Z be a solution of the instance. From Lemma 3.5, we know that a forbidden set C of G is disconnected by Z or $Z \cap C \neq \phi$. In the latter case, we know that Z contains a vertex $x \in C$ giving us one of the branched instances obtained by adding x into the solution.

Now we are in the case when C is disconnected by Z. Since Reduction rule 1 is applied exhaustively, the connected component containing C also contains some vertices in W. Since Z is a non-separating solution, W goes to exactly one connected component of $G \setminus Z$ and there exists some non-empty part of C that is not in this component. Hence, there exists some vertex $x \in C$ that gets disconnected from W by Z. From Lemma 3.7, we know that there is also a solution Z' which contains an important $x - W$ separator of size at most k in G. Since we have branched over all such $x - W$ important separators, we have correctly guessed on one such branch.

We now bound the running time. Since $|C| = \mathcal{O}(d)$, any forbidden set in G can be obtained via brute force in $n^{\mathcal{O}(d)}$ time. For each $i \in [k]$, we know that there are at most 4^i important separators of size $1 \leq i \leq k$ which can be enumerated using Lemma 3.6 in $\mathcal{O}(4^i \cdot i \cdot n)$ time. For the instance (G, k, W), if we branch on $v \in C$, k drops by 1 and if we branch on a $v - W$ separator of size i, k drops by i. Hence if $T(k)$ denotes the time taken for the instance (G, k, W), we get the recurrence relation $T(k) = \mathcal{O}(d)T(k-1) + \sum\limits_{i=1}^{k} 4^i T(k-i)$ upon solving with taking into account that d is a constant, we get that $T(k) = 2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. ◀

3.3 Solving general instances

We now solve general instances of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC using the algorithm for solving non-separating instances as a subroutine. We first focus on solving separating instances of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. We guess a subset $W_1 \subset W$ such that for a solution Z, W_1 is exactly the intersection of W with a connected component of $G \setminus Z$. For $W_2 = W \setminus W_1$, we are looking for a solution Z containing a $W_1 - W_2$ separator. Formally, let $W = W_1 \uplus W_2$ be a set of size $k+1$ which is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator. We look for a set $Z \subseteq V(G) \setminus W$ of size at most k such that Z is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator, Z contains a minimal (W_1, W_2)-separator X and W_1 occurs in a connected component of $G \setminus Z$.

From here on, we assume that the separating instance (G, k, W) of Disjoint $(\Pi_1, \Pi_2, \ldots,$ $\Pi_d)$-VDC is represented as (G, k, W_1, W_2) where $W = W_1 \uplus W_2$. We branch over all partitions of W into W_1 and W_2 which adds a factor of 2^{k+1} to the running time. We now introduce the notion of tight separator sequences and t-boundaried graphs which are used to design the algorithm.

3.3.1 Tight Separator Sequences

We first look at a type of $W_1 - W_2$ separators where the graph induced on the vertices reachable from W_1 satisfy the property as defined below.

▶ **Definition 3.9.** *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. *We call a $W_1 - W_2$ separator X in G* ***ℓ-good*** *if there exists a set K of size at most ℓ such that $K \cup X$ is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator for the graph $G[R[W_1, X]]$. Else we call it* ***ℓ-bad****.*

▶ **Lemma 3.10** (⋆)**.** *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and let X and Y be disjoint $W_1 - W_2$ separators in G such that X covers Y. If X is ℓ-good, then Y is also ℓ-good.*

▶ **Definition 3.11.** *Let (G, k, W_1, W_2) be an instance of* DISJOINT $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and let X and Y be $W_1 - W_2$ separators in G such that Y dominates X. Let ℓ be the smallest integer i for which X is i-good. If Y is ℓ-good, then we say that Y* ***well-dominates*** *X. If X is ℓ-good and there is no $Y \neq X$ which well-dominates X, then we call X* ***ℓ-important****.*

The following lemma allows us to focus on solutions containing an ℓ-important $W_1 - W_2$ separator for some appropriate value of ℓ.

▶ **Lemma 3.12.** *Let (G, k, W_1, W_2) be an instance of* DISJOINT $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and Z be a solution. Let $P \subseteq Z$ be a non-empty minimal $W_1 - W_2$ separator in G and let P' be a $W_1 - W_2$ separator in G well-dominating P. Then there is also a solution Z' for the instance containing P'.*

Proof. Let $Q = Z \cap R[W_1, P]$. Note that Q is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator for the graph $G[R[W_1, P]]$. Let $Q' \supseteq P'$ be a smallest $(\Pi_1, \Pi_2, \ldots, \Pi_d)$- modulator for the graph $G[R[W_1, P']]$ extending P'. We claim that $Z' = (Z \setminus Q) \cup Q'$ is a solution for (G, k, W_1, W_2). Since P' well-dominates P, $|Z'| \leq |Z|$. We now show that Z' is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator. Suppose not. Then there exists a forbidden subset C present in a connected component $\mathcal{X}$ of $G \setminus Z'$.

We first consider the case when $\mathcal{X}$ is disjoint from the set $Z \setminus Z'$. Then there is a component $\mathcal{H}$ in $G \setminus Z$ which contains $\mathcal{X}$ and hence C, contradicting that Z is a solution. We now consider the case when $\mathcal{X}$ intersects $Z \setminus Z'$. By definition of Z', $\mathcal{X}$ is contained in the set $R(W_1, P')$. Since $Z' \setminus Q'$ is disjoint from $R(W_1, P')$ and is separated from $R(W_1, P')$ by just P', we can conclude that $\mathcal{X}$ and hence C is contained in a single connected component of $G[R[W_1, P']] \setminus Q'$. But this contradicts that Q' is not a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator in the graph $G[R[W_1, P']]$. ◀

We now define the notion of tight separator sequence. It gives a natural way to partition the graph into parts with small *boundaries*. This helps us focus our problem on local parts of the graph which eases the task of solving it.

▶ **Definition 3.13.** *An $X - Y$* ***tight separator sequence of order k*** *of a graph G with $X, Y \subseteq V(G)$ is a set $\mathcal{H}$ of $X - Y$ separators such that every separator has size at most k, the separators are pairwise disjoint, for any pair of separators in the set, one covers another and the set is maximal with respect to the above properties.*

▶ **Lemma 3.14** ($\star$)**.** *Given a graph G, disjoint vertex sets X, Y and integer k, a tight separator sequence $\mathcal{H}$ of order k can be computed in $|V(G)|^{\mathcal{O}(1)}$ time.*

3.3.2 Boundaried graphs

▶ **Definition 3.15.** *A* ***t-boundaried graph*** *G be a graphs with t distinguished labelled vertices. We call the set of labelled vertices $\partial(G)$ the boundary of G and the vertices in $\partial(G)$ terminals. Let G_1 and G_2 be two t-boundaried graphs with the graphs $G_1[\partial(G_1)]$ and $G_2[\partial(G_2)]$ being isomorphic. Let $\mu : \partial(G_1) \to \partial(G_2)$ be a bijection which is an isomorphism of the graphs $G_1[\partial(G_1)]$ and $G_2[\partial(G_2)]$. We denote the graph $G_1 \otimes_\mu G_2$ as a t-boundaried graph obtained by the following gluing operation. We take the union of graphs G_1 and G_2 and identify each vertex $x \in \partial(G_1)$ with vertex $\mu(x) \in \partial(G_2)$. The t-boundary of the new graph is the set of vertices obtained by unifying.*

▶ **Definition 3.16.** *A* ***t-boundaried graph with an annotated set*** *is a t-boundaried graph with a second set of distinguished but unlabelled vertices disjoint from the boundary. The set of annotated vertices is denoted by $\Delta(G)$.*

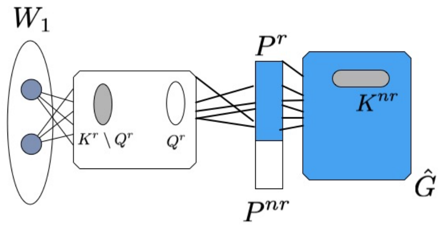

Figure 1 The graph G' obtained by gluing some $\hat{G} \in \mathcal{H}$ to $G[R[W_1, P]]$.

We now prove the following crucial lemma for giving the algorithm for solving separating instances of Disjoint $(\Pi_1, \Pi_2, \dots, \Pi_d)$-VDC. Suppose we are looking at an instance of Disjoint $(\Pi_1, \Pi_2, \dots, \Pi_d)$-VDC which has a solution Z. We know that the solution Z contains a minimal $W_1 - W_2$ separator Q which is incomparable to a given ℓ-good $W_1 - W_2$ separator P in G. Then some part of Z, say K, lies in the set $R[W_1, Q]$. We show that by carefully replacing parts outside of $R[W_1, P]$ with a small gadget, we can get a smaller graph G' which preserves the part of K inside the set $R[W_1, P]$. Also we show that there is some part of K lying outside the set $R[W_1, P]$ and hence the size of the solution in G' is strictly smaller than in G.

The gadget is loosely speaking a set obtained by a marking procedure which preserves all possible types of forbidden sets in the original graph. Let us look at a forbidden set C and a subset $C_i \subseteq C$ such that $G[C_i] = H_i \in \mathcal{F}_i$. Let us look at the subset $C_i' \subseteq C_i$ that lies outside $R(W_1, P)$, the corresponding graph being an induced subgraph of H_i. For all $i \in [d]$ and all possible induced subgraphs of H_i, we mark sets of vertices outside $R(W_1, P)$ such that there is a corresponding forbidden set C in the graph G whose intersections with $NR[W_1, P]$ are exactly these graphs. Let G' be the graph formed by removing the unmarked vertices outside $R(W_1, P)$. For any forbidden set C in G, we can replace parts of C_i outside $R(W_1, P)$ with the corresponding marked vertices and get a forbidden set C' in graph G'. This allows us to focus on the smaller graph G' and use the solution in G' to construct a solution in G.

▶ **Lemma 3.17** ($\star$). *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \dots, \Pi_d)$-VDC *and let Z be a solution. Let $Q \subseteq Z$ be a minimal $W_1 - W_2$ separator in the graph G. Let $K = Z \cap R[W_1, Q]$ with $\ell = |K \setminus Q|$. Let P be a minimal $W_1 - W_2$ separator in G which is disjoint from K and incomparable to Q. Let $Q^r = Q \cap R(W_1, P)$ and $Q^{nr} = Q \setminus Q^r$. Similarly, let $P^r = P \cap R(W_1, Q)$ and $P^{nr} = P \setminus P^r$. Let $K^r = K \cap R[W_1, P]$. Let $G_1 = G[R[W_1, P]]$ be a boundaried graph with P^r as the boundary.*

Then there exists a $|P^r|$-boundaried graph $\hat{G}$ which is $k^{\mathcal{O}(1)}$ in size with an annotated set of vertices, and a bijection $\mu : \partial(\hat{G}) \to P^r$ such that the glued graph $G' = G_1 \otimes_\mu \hat{G}$ has the following properties (see Figure 1).

1. *The set W_1 is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$-modulator for the graph G'.*
2. *The set Q^r is a $|K^r \setminus Q^r|$-good $W_1 - P^{nr}$ separator in the graph $G' \setminus \Delta(\hat{G})$.*
3. *For any Q' which is a $W_1 - P^{nr}$ separator in $G' \setminus \Delta(\hat{G})$ well-dominating Q^r in G', the set $Q' \cup Q^{nr}$ well dominates the $W_1 - W_2$ separator Q in G.*
4. *There is a a family $\mathcal{H}$ of boundaried graphs with an annotated set of vertices such that $\mathcal{H}$ contains $\hat{G}$, has size bounded by $2^{k^{\mathcal{O}(1)}}$ and can be computed in $2^{k^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$.*

3.3.3 Algorithm for general instances

The following lemma focuses on the particular case when W_1 and W_2 are already disconnected in G.

▶ **Lemma 3.18** ($\star$). *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *where W_1 and W_2 are in distinct components of G. Let Z be its solution such that W_1 exactly occurs in a connected component of $G \setminus Z$. Also let $R(W_1)$ be the set of vertices reachable from W_1 in G. Let $Z' = Z \cap R(W_1)$. Then $(G[R(W_1)], |Z'|, W_1)$ is a non-separating YES-instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and conversely for any non-separating solution Z'' for $(G[R(W_1)], |Z'|, W_1)$, the set $\hat{Z} = (Z \setminus Z') \cup Z''$ is a solution for the original instance such that W_1 exactly occurs in a connected component of $G \setminus Z''$.*

We have the following reduction rule.

▶ **Reduction Rule 2.** *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *where W_1 and W_2 are disconnected in G. Compute a non-separating solution Z' for the instance (G', k', W_1) where $G' = G[R(W_1)]$ and k' is the least integer $i \leq k$ for which (G', i, W_1) is a YES-instance. Delete Z' and return the instance $(G \setminus Z', k - |Z'|, W_2)$.*

The safeness of Reduction Rule 2 comes from Lemma 3.18. The running time for the reduction is $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ which comes from that of the algorithm in Lemma 3.8. Henceforth, we assume that Reduction Rule 2 is no longer applicable. We know that every solution Z of (G, k, W_1, W_2) contains an ℓ-good non-empty $W_1 - W_2$ separator X in the graph G. Let us denote Main-Algorithm(G, k, W_1, W_2) as the main algorithm procedure. We now describe a subroutine Branching-Set$((G, k, W_1, W_2), \lambda, \ell)$ which is used in Main-Algorithm where $0 \leq \ell \leq k$ and $1 \leq \lambda \leq k$.

▶ **Lemma 3.19.** *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and let $0 \leq \ell \leq k$ and $1 \leq \lambda \leq k$. There is an algorithm Branching-Set$((G, k, W_1, W_2), \lambda, \ell)$ that returns a set $\mathcal{R}$ containing at most $2^{k^{\mathcal{O}(1)}}$ vertices such that for every solution Z of the given instance containing an ℓ-important $W_1 - W_2$ separator X of size at most λ in G, the set $\mathcal{R}$ intersects Z.*

Proof. Since Reduction Rule 2 is applied exhaustively, there is a $W_1 - W_2$ path in the graph G. If there is no $W_1 - W_2$ separator of size λ in the graph G, we declare the tuple invalid. Else we execute Lemma 3.14 to obtain a tight $W_1 - W_2$ separator sequence $\mathcal{I}$ of order λ. We then partition $\mathcal{I}$ into ℓ-good and ℓ-bad separators. For this we need a procedure to check whether a given separator P is ℓ-good or not. If both $\lambda, \ell < k$, we do so by recursively calling the main algorithm procedure Main-Algorithm$(G[R[W_1, P]], \ell, W_1, P)$. The recursive call is possible as ℓ is strictly less than k. We note that since $\lambda \geq 1$, the cardinality of P is non-zero. From the definition of ℓ-good separators, we can conclude that $\ell < k$ as the solution set in $G[R[W_1, P]]$ is at most k and it contains non-empty set P as its subset.

Let P_1 be component maximal among all the good separators in $\mathcal{I}$ if any exists and P_2 be component minimal among all the bad separators in $\mathcal{I}$ if any exists. We set $\mathcal{R} := P_1 \cup P_2$. For $i \in \{1, 2\}$, we do the following.

We execute the algorithm of Lemma 3.17, Claim 4 to compute a family $\mathcal{H}$ of boundaried graphs with an annotated set of vertices. Then for every choice of $P_i^r \subseteq P_i$, for every instance $\hat{G} \in \mathcal{H}$ with $|P_i^r|$ terminals and every possible bijection $\delta : \partial(\hat{G}) \to P_i^r$, we construct the glued graph $G_{P_i^r, \delta} = G[R[W_1, P_i]] \otimes_\delta \hat{G}$, where the boundary of $G[R[W_1, P_i]]$ is P_i^r.We then recursively call Branching-Set$((G_{P_i^r, \delta} \setminus \tilde{S}, k - j, W_1, P_i \setminus P_i^r), \lambda', \ell')$ for every $0 \leq \lambda' < \lambda$, $1 \leq j \leq k - 1$ and $0 \leq \ell' \leq \ell$, where $\tilde{S}$ is the set of annotated vertices in $\hat{G}$. We add the union of all the vertices returned by these recursive instances to $\mathcal{R}$ and return the resulting set.

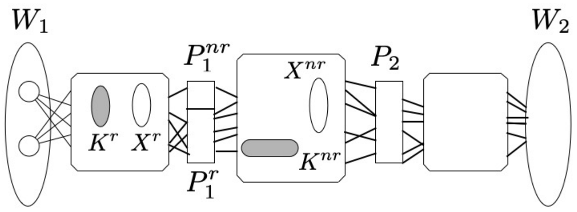

Figure 2 The case where X is incomparable with P_1.

This completes the description of the algorithm. We now proceed to the proof of correctness.

Correctness: We prove by induction on λ. We first consider the base case when $\lambda = 1$ where there is a $W_1 - W_2$ ℓ-good separator $X \subseteq Z$ of size one. Since X has size one, it cannot be incomparable with the separator P_1. Hence either X is equal to P_1 or is covered by P_1. In either case, we are correct as P_1 is contained in $\mathcal{R}$. We now assume that the algorithm correctly runs for all tuples where $\lambda < \hat{\lambda}$ for some $\hat{\lambda} \geq 2$. We now look at the case when the algorithm runs on a tuple with $\lambda = \hat{\lambda}$.

Let Z be a solution for the instance containing an ℓ-important separator X. If X intersects $P_1 \cup P_2$ we are done as $\mathcal{R}$ intersects X. Hence we assume that X is disjoint from $P_1 \cup P_2$. Suppose X is covered by P_1. Then we conclude that P_1 well-dominates X contradicting that X is ℓ-important.

By Lemma 3.10, since X is ℓ-good and P_2 is not, X cannot cover P_2. Suppose X covers P_1 and itself is covered by P_2. Then X must be contained in the maximal tight separator sequence $\mathcal{I}$ contradicting that P_1 is component maximal.

Finally we are left with the case where X is incomparable with P_1 or P_2 if P_1 does not exist. Without loss of generality, assume X is incomparable with P_1. The argument in the case when P_1 does not exist follows by simply replacing P_1 with P_2 in the proof.

Let $K \subseteq Z$ be a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator for the graph $G[R[W_1, X]]$ extending X. If $P_1 \cap K$ is empty, we have that $P_1 \cap Z$ is empty. Since P_1 is contained in $\mathcal{R}$, the algorithm is correct as $\mathcal{R}$ intersects Z. Hence assume that P_1 and K are disjoint.

Let $X^r = R(W_1, P_1) \cap X$ and $X^{nr} = X \setminus X^r$. Similarly, define $P_1^r = R(W_1, X) \cap P_1$ and $P_1^{nr} = P_1 \setminus P_1^r$. Since X and P_1, the sets X^r, X^{nr}, P_1^r and P_1^{nr} are all non-empty. Let $K^r = K \cap R(W_1, P_1)$. If there is a vertex in P_1^r that is not in the same connected component as W_1 in $G \setminus Z$, then as X separates P_1^r from W_2, we can conclude that $\mathcal{R}$ contains a vertex which is separated from W by Z implying that the algorithm is correct. Hence assume that P_1^r is contained in the same connected component as W_1 in the graph $G \setminus Z$.

We observe that the sets defined above satisfy the conditions for Lemma 3.17 with $P = P_1$ and $Q = X$. Therefore there exists a $|P_1^r|$-boundaried graph $\hat{G}$ with an annotated set $\tilde{S}$ and an appropriate bijection $\mu : \partial(\hat{G}) \rightarrow P_1^r$ with the properties claimed in the statement of Lemma 3.17. Now consider the recursive instance $\langle (G_{P_i^r,\delta} \setminus \tilde{S}, k_1, W_1, P_i^{nr}), \lambda', \ell' \rangle$ where $G_{P_i^r,\delta}$ is the graph obtained by gluing together $G[R[W_1, P_1]]$ and $\hat{G}$ via a bijection μ , $\lambda' = |X^r|, k_1 = |K^r|$ and $\ell' = |K^r|$.

To apply induction hypothesis on the above tuple, we first show that the tuple is valid. We show this by showing that $(G_{P_i^r,\delta} \setminus \tilde{S}, k_1, W_1, P_1^{nr})$ is a valid instance of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. For this we need that $W_1 \cup P_1^{nr}$ is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator for the graph $G_{P_i^r,\delta} \setminus \tilde{S}$ which is true from Lemma 3.17, Claim 1. Hence the tuple is valid and we can apply the induction hypothesis.

Since X is ℓ-important, from Lemma 3.17, Claims 2 and 3, it follows that X^r must also be k_1-important in the graph $G_{P_i^r,\delta} \setminus \tilde{S}$. By induction hypothesis, the tuple returns a set $\mathcal{R}'$ which intersects K^r. Since $K^r \subseteq Z$, we can conclude that $\mathcal{R}'$ intersects Z as well.

Bounding the set $\mathcal{R}$: We have the value of λ dropping at every level of the search tree. Since $\lambda \leq k$, the depth of the search tree is bounded by k. The number of branches at each node is at most $k^3 \cdot k! \cdot 2^{k^{\mathcal{O}(1)}}$ (k^3 for choice of λ', j and ℓ', $k!$ for the choice of the bijection δ and $2^{k^{\mathcal{O}(1)}}$ for the size of $\mathcal{H}$). Since, at each internal node, we add at most $2k$ vertices (corresponding to $P_1 \cup P_2$), we can conclude that the size of $\mathcal{R}$ is bounded by $2^{k^{\mathcal{O}(1)}}$. ◀

We now describe the details of Main-Algorithm procedure.

▶ **Lemma 3.20.** *There is a procedure Main-Algorithm that given an instance (G, k, W_1, W_2) of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC*, runs in $2^{k^{O(1)}} n^{O(1)}$ and either computes a solution for the instance containing a $W_1 - W_2$ separator or concludes that no such solution exists.*

Proof. Initially, if Reduction Rule 2 is applicable (this is the case when the size of the minimum cut λ in G is zero), we use it to reduce the instance. Hence we can assume that the $W_1 - W_2$ separator is non-empty. For every $0 \leq \ell' \leq k$ and $1 \leq \lambda' \leq k$, we invoke the algorithm Branching-Set$((G, k, W_1, W_2), \lambda', \ell')$ from Lemma 3.19 to compute a set $\mathcal{R}_{\lambda',\ell'}$. We define $\mathcal{R}$ as the union of the sets $\mathcal{R}_{\lambda',\ell'}$ for all possible values of λ' and ℓ'. After this, we simply branch on every vertex v of R adding v to the solution creating a new instance $(G - v, k - 1, W_1, W_2)$ of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. If $k < 0$, we return NO. If Reduction Rule 2 applies, we use it to reduce the instance. If this results in a non-separating instance with $W = W_1 \cup W_2$, we apply the algorithm in Lemma 3.8 to solve the instance. Else we recursively run Main-Algorithm on the new instance.

We now bound the running time $T(k)$ for Main-Algorithm. The depth of the branching tree is bounded by k and the branching factor at each node is $|\mathcal{R}| \leq 2^{k^{\mathcal{O}(1)}}$. The time taken at each node is dominated by the time taken for the procedure Branching-Set. Let $Q(k)$ denote the time taken for Branching-Set. We have $T(k) = 2^{k^{\mathcal{O}(1)}} T(k-1) + Q(k)$. Let us focus on the search tree for Branching-Set. In Lemma 3.19, we proved that the depth of the tree is bounded by k and the branching factor is bounded by $2^{k^{\mathcal{O}(1)}}$. The time spent at each node is dominated by algorithm of Lemma 3.17 and that of the sub-instances of Main-Algorithm called at the node with strictly smaller values of k which is bounded by $2^{k^{\mathcal{O}(1)}} n^{O(1)} + T(k-1)$. Hence overall we have $Q(k) = 2^{k^{\mathcal{O}(1)}} Q(k-1) + 2^{k^{\mathcal{O}(1)}} n^{O(1)} + T(k-1)$. Substituting $Q(k)$ in the recurrence relation for the running time of Main-Algorithm and simplifying, we have $T(k) = \sum_{i=1}^{k} 2^{ik^{\mathcal{O}(1)}} n^{O(1)} T(k-i) \leq k \cdot 2^{k \cdot k^{\mathcal{O}(1)}} n^{O(1)} T(k-1)$ on solving we get $T(k) = 2^{k^{\mathcal{O}(1)}} n^{O(1)}$.

The correctness follows from the correctness of Lemma 3.19, of Reduction Rule 2 and Lemma 3.8. ◀

▶ **Lemma 3.21.** Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *can be solved in $2^{k^{\mathcal{O}(1)}} n^{O(1)}$ time.*

Proof. Let (G, k, W) be the instance of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. We first apply Lemma 3.8 to see if there is a non-separating solution for the instance. If not, we branch over all $W_1 \subset W$ and for each such choice of W_1, apply Lemma 3.20 to check if $(G, k, W_1, W_2 = W \setminus W_1)$ has a solution containing a $W_1 - W_2$ separator. The correctness and running time follows from those of Lemma 3.8 and Lemma 3.20. ◀

▶ **Theorem 3.22.** $(\Pi_1, \Pi_2, \dots, \Pi_d)$ Vertex Deletion *can be solved in* $2^{k^{\mathcal{O}(1)}} n^{O(1)}$ *time.*

Proof. As mentioned in Section 3.1, the time taken to solve $(\Pi_1, \Pi_2, \dots, \Pi_d)$ Vertex Deletion is $2^{k+1} \cdot 2^{k^{\mathcal{O}(1)}} n^{O(1)} = 2^{k^{\mathcal{O}(1)}} n^{O(1)}$. ◀

4 Finite Forbidden Set with Paths

We observe that several natural graph classes (like cluster graphs, edgeless graphs, P_5-free graphs) contain a path in their forbidden sets. In this section, we develop significantly faster FPT algorithms for the vertex deletion problem if each connected component of the resulting graph belongs to one of two graph classes and the forbidden set of one of them contains a path.

Deletion to Π_1 and Π_2 with Path
Input: An undirected graph G, and an integer k. Furthermore, for a fixed i, an induced path P_i is a forbidden set for Π_1.
Goal: Does G have a set S of at most k vertices such that every connected component of $G - S$ is either in Π_1 or in Π_2?

Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be the finite forbidden sets for the graph classes Π_1 and Π_2 respectively. Let d_1 be the size of a maximum sized finite forbidden set in $\mathcal{F}_1$ and d_2 be the size of a maximum sized forbidden set in $\mathcal{F}_2$. We first recall the graph operation called *gluing* that was defined in Section 3.3. We spell it out for our purpose.

Gluing Operation. Let G_1 and G_2 be graphs with H being an induced subgraph of both of them. Let $S_1 \subseteq V(G_1)$ and $S_2 \subseteq V(G_2)$ be such that $G_1[S_1] = G_2[S_2] = H$. Let f be an automorphism of H, i.e. an isomorphism from H onto H, that maps vertices of H to vertices of H. We define a collection of graphs $G \in Glue(G_1, G_2, H, S_1, S_2, f)$ with $V(G) = V(G_1) \cup V(G_2) \setminus S_2$ whose edge set $E(G)$ is as follows. We treat $V(G)$ as containing all vertices of G_1, and containing vertices of $V(G_2) \setminus S_2$. I.e. we identify vertices of S_1 and S_2 using the function f.

(1) For any pair u, v of vertices in $V(G_1)$ in $V(G)$, we add an edge if and only if $uv \in E(G_1)$, **(2)** for any pair u, v of vertices in $V(G_2) \setminus S_2$, we add an edge if and only if $uv \in E(G_2)$, **(3)** for a pair u, v of vertices where $u \in S_1$ and $v \in V(G_2) \setminus S_2$, we add the edge if and only if $f(u)v \in E(G_2)$, **(4)** for a pair u, v of vertices where $u \in V(G_1) \setminus S_1$ and $v \in V(G_2) \setminus S_2$, we have two choices. In the first choice, we add edge uv to G. In the second choice, we do not add edge uv to G.

Note that this construction provides a collection of graphs based on the last bullet point above, even for a fixed f, and for a fixed S_1 and S_2. Now towards the deletion algorithm for two classes where one of them contains a path, we first construct the following family of graphs from $\mathcal{F}_1$ and $\mathcal{F}_2$.

Construction of a new family $\mathcal{F}$ from $\mathcal{F}_1$ and $\mathcal{F}_2$. For every $F_1 \in \mathcal{F}_1, F_2 \in \mathcal{F}_2$, we construct a collection of graph $\mathcal{F}'$ by $Glue(F_1, F_2, S_1, S_2, H, f)$ for every graph H which is a subgraph of both F_1 and F_2 and for every subsets S_1 and S_2 of vertices of F_1 and F_2 respectively such that $F_1[S_1] = H = F_2[S_2]$ and for every automorphism f of H. In addition, we construct $\mathcal{F}'', \mathcal{F}'''$ as follows. The set $\mathcal{F}''$ contains all graphs formed by disjoint union of $F_1 \in \mathcal{F}_1, F_2 \in \mathcal{F}_2$ and some subset of edges between them. The set $\mathcal{F}'''$ contains all graphs formed by disjoint union of $F_1 \in \mathcal{F}_1, F_2 \in \mathcal{F}_2$ and a path P of length at most i (with $i - 1$ additional vertices) starting from a vertex in F_1 and ending at a vertex in F_2. We denote $\mathcal{F} = \mathcal{F}' \cup \mathcal{F}'' \cup \mathcal{F}'''$.

▶ **Lemma 4.1** (⋆). *Let Π_1, Π_2 be two graph classes such that a graph $G_1 \in \Pi_1$ has no induced subgraph from $\mathcal{F}_1$, and a graph $G_2 \in \Pi_2$ has no induced subgraph from $\mathcal{F}_2$. Let d_1 be the maximum number of vertices in any graph in $\mathcal{F}_1$ and d_2 be the maximum number of vertices in any graph in $\mathcal{F}_2$. Furthermore let $\mathcal{F}_1$ contains a path of length i. Construct the set $\mathcal{F}$ from $\mathcal{F}_1$ and $\mathcal{F}_2$ as described by "gluing operation" just before the lemma. Let Π be a class of graphs such that $G \in \Pi$ if and only if G has no induced subgraph present in $\mathcal{F}$. Then, every connected component of G is either in Π_1 or in Π_2 if and only if $G \in \Pi$. Furthermore, the maximum number of vertices in a set in $\mathcal{F}$ is at most $d_1 + d_2 + i$.*

▶ **Theorem 4.2** (⋆). *Let d_1 be the maximum size of an obstruction in $\mathcal{F}_1$ and d_2 be the maximum size of an obstruction in $\mathcal{F}_2$ for* Deletion to Π_1 and Π_2 with Path *problem. Then,* Deletion to Π_1 and Π_2 with Path *admits $(d_1 + d_2 + i)^k n^{\mathcal{O}(1)}$ time algorithm. It also admits $(d_1 + d_2 + i)$ factor approximation algorithm, and a polynomial sized kernel.*

Proof (Sketch). Given an instance (G, k) of Deletion to Π_1 and Π_2 with Path problem, we use "gluing operation" described above to construct a finite obstruction family $\mathcal{F}$. From Lemma 4.1, all obstructions in $\mathcal{F}$ have size at most $d_1 + d_2 + i$. We can prove that Π_1 or Π_2 Deletion is an instance of an implicit $d_1 + \hat{d} + 2 + i$-Hitting Set problem [5]. We can find induced graph H in G which is isomorphic to any forbidden set in $\mathcal{F}$ in polynomial time. By branching over the vertices of H we get an FPT algorithm running in time $(d_1 + d_2 + i)^k n^{\mathcal{O}(1)}$. We can get a $d_1 + d_2 + i$ factor approximation algorithm by greedily adding all vertices of H to the solution as usually done in implicit $(d_1 + d_2 + i)$-Hitting Set problems. We can use Sunflower Lemma [5, 7] to get a polynomial kernel for Π_1 or Π_2 Deletion. ◀

5 Deletion to Trees and Cliques

In this section, we consider the problem when Π_1 is the set of all cliques, and Π_2 is the set of all trees. That is, the problem is to delete k vertices so that in the resulting graph, each connected component is a tree or a clique.

Trees and Cliques Deletion Set	**Parameter:** k
Input: An undirected graph G, and an integer k	
Question: Does G have a set S of at most k vertices such that every connected component of $G - S$ is either a tree or a clique?	

We call a subset $S \subseteq V(G)$ a *trees-and-cliques deletion set* if $G \setminus S$ is such that every connected component of $G - S$ is either a tree or a clique. Note that when each component is a tree, the deletion problem is precisely the Feedback Vertex Set problem, with the resulting class of all acyclic graphs which have the set of all cycles as the (infinite) forbidden set. This has an FPT algorithm with the best runtime $O^*(3.618^k)$ [11]. See also [19] for the special deletion problem where we want the resulting graph to be a tree. When each connected component is a clique, the deletion problem is precisely the Cluster Vertex Deletion problem, with the resulting class of cluster graphs contains the graphs forbidding P_3s, paths on three vertices. For our problem, as the forbidden set for one of the graph classes is infinite, the fixed-parameter tractability does not follow from our Theorem 3.22 or Theorem 4.2.

In this section, we describe a $\mathcal{O}^*(4^k)$ time algorithm for Trees and Cliques Deletion Set problem. Recall that a *block graph* is a graph whose biconnected components are cliques. If every connected component of a graph is a tree or a clique, then clearly it is a block graph. First, we precisely characterize such block graphs whose components are trees or cliques. Recall that a paw is a graph on 4 vertices that contains a triangle and the fourth vertex is adjacent only to one vertex of the triangle. We have the following two lemmas.

▶ **Lemma 5.1.** *Let G be a block graph. Then, G is paw-free if and only if every connected component of G is either a tree or a clique.*

Proof. Suppose that every connected component of G is either a tree or a clique. As a paw has a cycle and is not a clique, G can not contain a paw. Conversely suppose G does not contain a paw, but one of its components C, is neither a tree or a clique. Then, the component must have a cycle, and a nonadjacent pair of vertices. But as G is a block graph, every biconnected component of G, and hence of C must be a clique. As C is not a clique, C has at least two biconnected components B_1 and B_2 that are cliques intersecting at a (cut) vertex. Let B_1 be the component containing a cycle. Then B_1 contains a triangle, B_2 contains an edge, and B_1 and B_2 share a vertex. This triangle and the edge form a paw, which is a contradiction. ◀

▶ **Lemma 5.2** ([2]). *A graph G is a block graph if and only if it has no induced cycles of length at least four and no induced D_4 (i.e., $K_4 - e$, or a diamond).*

The following corollary follows from the above lemmas.

▶ **Corollary 5.3.** *Every connected component of a graph G is either a tree or a clique if and only if it has no cycle of length at least four or a diamond or paw as induced subgraphs.*

Now, we will first focus on the following restricted version of Trees and Cliques Deletion Set problem where we assume that the graph has no C_4 (a cycle of length four), D_4 or a paw as an induced subgraph.

Restricted Trees and Cliques Deletion Set	**Parameter:** k
Input: A connected undirected graph G without C_4, D_4 and paw as induced subgraphs, and an integer k	
Question: Does G have a set S of at most k vertices such that every connected component of $G - S$ is either a tree or a clique?	

Now we give a fixed-parameter tractable algorithm for Restricted Trees and Cliques Deletion Set. We have the following lemma from [1].

▶ **Lemma 5.4** ([1]). *Let G be a graph that does not contain C_4 or D_4 as an induced subgraph. Then **(1)** any pair of maximal cliques of G intersects in at most one vertex, and **(2)** the number of maximal cliques in G is at most n^2.*

Let $\mathcal{C}$ denote the set of all maximal cliques of G. A vertex $v \in V(G)$ is called *external* if it is part of at least two maximal cliques of $\mathcal{C}$. We construct an auxiliary bipartite graph $\hat{G}$ from G with $V(\hat{G}) = V(G) \uplus V_{\mathcal{C}}$ where $V_{\mathcal{C}}$ has a vertex v_c for every $c \in \mathcal{C}$. In the graph $\hat{G}$, we add an edge from a vertex $v \in V(G)$ to a vertex $v_c \in V_{\mathcal{C}}$ if and only if v is one of the external vertices of the clique $c \in \mathcal{C}$. We prove the following lemma which is similar to Lemma 7 in [1].

▶ **Lemma 5.5** ($\star$). *Let G be a graph without C_4, D_4 and paw as induced subgraphs and $S \subseteq V(G)$. Then S is a trees-and-cliques deletion set of G if and only if $\hat{G} \setminus S$ is acyclic.*

Now we consider the weighted feedback vertex set problem

Weighted Feedback Vertex Set	**Parameter:** k
Input: A weighted undirected graph $G = (V, E)$ with $w : V(G) \to \mathbb{N}$, and an integer k.	
Question: Find a set $S \subseteq V(G)$ of minimum weight that contains at most k vertices.	

that has an $O^*(3.618^k)$ algorithm [1].

Hence to solve the Restricted Trees and Cliques Deletion Set, we can simply form the auxiliary graph as described in Lemma 5.5. Then we define a weight function on the vertices of $\hat{G}$ as those vertices of G taking weight 1, and those vertices of the maximal cliques taking value $k+1$. Then we apply the algorithm for weighted feedback vertex set to check whether the minimum weight of, a feedback vertex set of (the weighted graph) $\hat{G}$ containing at most k vertices, is at most k. Thus we have

▶ **Lemma 5.6.** Restricted Trees and Cliques Deletion Set *can be solved in* $\mathcal{O}^*(3.618^k)$ *time.*

Now by branching on vertices of C_4s, D_4s and paws and then applying the algorithm of Lemma 5.6, we obtain the following theorem.

▶ **Theorem 5.7.** *Given a graph G on n vertices, we can determine in $\mathcal{O}^*(4^k)$ time whether G has at most k vertices whose deletion results in a graph where every connected component is a tree or a clique.*

Proof. Let $S \subseteq V(G)$ be a set of vertices such that every connected component of $G - S$ is either a tree or a clique. Observe that $G - S$ cannot contain a C_4, diamond or paw as induced subgraph. Hence, S must intersect all induced C_4, all induced diamonds and all induced paws in G. As long as we find a set of four vertices A such that $G[A]$ induces a C_4, or a paw, or a diamond, we branch on every vertex $v \in A$, and solve recursively the instance $(G - \{v\}, k-1)$. If one of these branches returns a solution X, we return $X \cup \{v\}$ as a solution of G. Otherwise, we return that (G, k) is a no-instance. After G has no four vertices that induces a diamond, or a C_4, or a paw, then, we do not make any further recursive call. We invoke Lemma 5.6 to apply the algorithm for Restricted Trees and Cliques Deletion Set and return the output of the algorithm. Since, the algorithm for Restricted Trees and Cliques Deletion Set takes $\mathcal{O}^*(3.618^k)$ time and we branch on at most k C_4, diamonds and paws, our algorithm takes $\mathcal{O}^*(4^k)$ time. This completes the proof. ◀

6 Conclusion

We have initiated a study on vertex deletion problems to scattered graph classes and showed that when there are a finite number of graph classes each characterized by a finite forbidden set, the problem is fixed-parameter tractable. The existence of a polynomial kernel for this case is a natural open problem. Other open problems include obtaining improved algorithms at least for special cases of finite classes and investigating other scattered graph classes when some of them have infinite forbidden sets.

References

1 Akanksha Agrawal, Sudeshna Kolay, Daniel Lokshtanov, and Saket Saurabh. A faster FPT algorithm and a smaller kernel for block graph vertex deletion. In *LATIN 2016: Theoretical Informatics*, pages 1–13. Springer, 2016.

2 Andreas Brandstadt, Jeremy P Spinrad, et al. *Graph classes: a survey*, volume 3. Siam, 1999.

3 Leizhen Cai. Fixed-Parameter Tractability of Graph Modification Problems for Hereditary Properties. *Inf. Process. Lett.*, 58(4):171–176, 1996.

4 Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.

5 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

6 Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 3. Springer, 2015.

7 Walter A. Deuber, Paul Erdös, David S. Gunderson, Alexandr V. Kostochka, and A. G. Meyer. Intersection Statements for Systems of Sets. *J. Comb. Theory, Ser. A*, 79(1):118–132, 1997.

8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

9 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. *ACM Trans. Algorithms*, 13(2):29:1–29:32, 2017.

10 Pinar Heggernes, Pim van 't Hof, Bart M. P. Jansen, Stefan Kratsch, and Yngve Villanger. Parameterized complexity of vertex deletion into perfect graph classes. *Theor. Comput. Sci.*, 511:172–180, 2013.

11 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic Feedback Vertex Set. *Inf. Process. Lett.*, 114(10):556–560, 2014.

12 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.

13 Daniel Lokshtanov. Wheel-free deletion is W[2]-hard. In *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, pages 141–147, 2008.

14 Daniel Lokshtanov and MS Ramanujan. Parameterized tractability of multiway cut with parity constraints. In *International Colloquium on Automata, Languages, and Programming*, pages 750–761. Springer, 2012.

15 Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

16 Geevarghese Philip, Ashutosh Rai, and Saket Saurabh. Generalized pseudoforest deletion: Algorithms and uniform kernel. *SIAM J. Discrete Math.*, 32(2):882–901, 2018.

17 Ashutosh Rai and M. S. Ramanujan. Strong parameterized deletion: Bipartite graphs. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, pages 21:1–21:14, 2016.

18 Ashutosh Rai and Saket Saurabh. Bivariate complexity analysis of almost forest deletion. *Theor. Comput. Sci.*, 708:18–33, 2018.

19 Venkatesh Raman, Saket Saurabh, and Ondrej Suchý. An FPT algorithm for Tree Deletion Set. *J. Graph Algorithms Appl.*, 17(6):615–628, 2013.

20 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM J. Comput.*, 10(2):310–327, 1981.