Advancing the Algorithmic Tool-kit for Parameterized Cut Problems

By

Roohani Sharma

MATH10201504012

The Institute of Mathematical Sciences, Chennai

A thesis submitted to the

Board of Studies in Mathematical Sciences

In partial fulfillment of requirements

for the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

August, 2020

Homi Bhabha National Institute

Recommendations of the Viva Voce Committee

As members of the Viva Voce Committee, we certify that we have read the dissertation prepared by Roohani Sharma entitled "Advancing the Algorithmic Toolkit for Parameterized Cut Problems" and recommend that it may be accepted as fulfilling the thesis requirement for the award of Degree of Doctor of Philosophy.

Chairman - Venkatesh Raman — Date: 13 August 2020

Guide/Convenor - Saket Saurabh — Date: August 13, 2020

Examiner - Manu Basavaraju — Date: 13 August 2020

Member 1 - Meena Mahajan — Date: 13 Aug 2020

Member 2 - Vikram Sharma — Date: 13 August 2020

Member 3 - Geevarghese Philip — Date: August 13, 2020

Final approval and acceptance of this thesis is contingent upon the candidate's submission of the final copies of the thesis to HBNI.

I hereby certify that I have read this thesis prepared under my direction and recommend that it may be accepted as fulfilling the thesis requirement.

Date: August 13, 2020

Place: Chennai

Saket Saurabh (Guide)

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at Homi Bhabha National Institute (HBNI) and is deposited in the Library to be made available to borrowers under rules of the HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the Competent Authority of HBNI when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

Roohani Sharma

Contents

Summary **i**

List of Figures **v**

I Introduction **1**

1 About the Thesis **3**

1.1 Digraph Cut Problems: Advancing the Otherwise Limited Tool-Kit . 5

1.1.1 Towards a Polynomial Kernel for DFV/AS 6

1.1.1.1 DFAS: Restricting the Input 8

1.1.1.2 DFVS: Enriching the Parameter 9

1.1.1.3 DFVS: Restricting the Output 10

1.1.2 Sub-Exponential FPT Algorithms Beyond Tournaments 11

1.2 Undirected Cut Problems: Exploring and Extending the Rich Tool-Kit 14

1.3 Towards the Re-Usability of Algorithms 15

1.4 Organization of the Thesis 17

2 Preliminaries **19**

2.1 Parameterized Complexity and Kernelization 19

2.2 Notations and Basic Definitions 20

2.3 Degeneracy of Graphs 23

2.4 Graph (Important) Separators 23
2.5 Treewidth 24
2.6 Branching Algorithms 26

II Polynomial Kernels For Directed Feedback Vertex/Arc Set And Its Variants 27

3 Kernel for DFAS on Bounded Independence Number Digraphs via Reachability Preserving Fault-Tolerant Subgraphs 29

3.1 Reachability Preserving Fault-Tolerant Subgraphs 29
3.2 Introducing Cut Preserving Sets 32
3.2.1 Computing k-Cut Preserving Sets: The Ideas 33
3.3 Prelude to the Technical Details 34
3.4 Finding Small k-Cut Preserving Sets 35
3.4.1 Finding a Small k-Cut Preserving Set for a Pair with a Large Flow 41
3.4.2 Finding a Small k-Cut Preserving Set for a Pair in a Tournament 41
3.4.3 Finding a small k-Cut preserving set for a pair in a $D \in \mathcal{D}_\alpha$. 44
3.4.4 k-Cut Preserving Sets for a Set of Vertices 46
3.5 Applications of the k-Cut Preserving Lemma 47
3.5.1 Fault-Tolerant (S, S)-Reachability 47
3.5.2 Kernel for DFAS on $\mathcal{D}_\alpha$ 50

4 Kernel for DEOCT on Bounded Independence Number Digraphs via Parity Reachability Preserving Fault-Tolerant Subgraphs 53

4.1 NP-hardness of DEOCT on Tournaments 54
4.2 Parity Preserving Fault-Tolerant Subgraphs 57
4.3 Introducing Parity Preserving Sets 58

4.3.1 Preserving Path Length Modulo p: The Ideas 59
4.4 Prelude to the Technical Details 59
4.5 Computing Parity Preserving Sets 60
4.6 Applications of Parity Preserving Sets 63
4.6.1 Fault-Tolerance (S, S)-Parity Reachability 63
4.6.2 Kernel for mod(p)-DCT in $\mathcal{D}_\alpha$ 65

5 Polynomial Kernel for DFVS Parameterized by the Solution Size Plus the Size of a Treewidth-η Modulator 67

5.1 Overview of the Kernelization Algorithm 69
5.2 Prelude to the Technical Details 72
5.3 Decomposing the Graph 72
5.4 Reducing Each Part: k-DFVS Representative Marking 76
5.4.1 Revisiting the Relation with the Skew Multicut Problem . . . 77
5.4.2 Computing Solutions for All Instances of Skew Multicut . 80
5.4.2.1 Proof of the k-SMC Representative Marking Lemma 84
5.4.2.2 Proof of the k-DFVS Representative Marking Lemma 87
5.5 Reduction Rules 88
5.5.1 Limiting the Interaction Between M and Z 89
5.5.2 Protrusion Replacement and Proof of the Main Theorem . . . 91
5.6 Towards the Proof of the Protrusion Replacer Lemma 93
5.6.1 Counting Monadic Second Order Logic and its Properties . . . 94
5.6.2 Boundaried Graphs 95
5.6.3 Finite Integer Index 95
5.6.4 Proof of the Protrusion Replacer Lemma 97

6 Kernel for Deletion to Out-Forest 99

6.1 Improved Kernel for Out-Forest Vertex Deletion Set 100
6.1.1 Some Prerequisites 100
6.1.2 Approximation Algorithm for Out-Forest Vertex Deletion Set 101
6.1.3 Kernelization Algorithm for Out-Forest Vertex Deletion Set 102
6.2 Kernel Lower Bound for Out-Forest Vertex Deletion Set . . . 106

7 Kernel for Deletion to Pumpkin 109
7.1 Outline of the Kernelization Algorithm for Pumpkin Vertex Deletion Set 109
7.2 The Simplification Phase 110
7.3 The Marking Approach 115

8 Faster FPT Algorithms for Deletion to Out-Tree, Out-Forest and Pumpkin 121
8.1 FPT Algorithm for Pumpkin Vertex Deletion Set 123
8.2 FPT Algorithm for Out-Tree Vertex Deletion Set 133
8.3 FPT Algorithm for Out-Forest Vertex Deletion Set 144

9 Kernel for Deletion to Bounded Treewidth DAGs 155

III Sub-Exponentiality Beyond Tournaments 161

10 Sub-Exponential FPT Algorithms for DFAS, Directed Cutwidth and Optimal Linear Arrangement in Digraphs of Bounded Independence Number 163
10.1 Our Methods and k-Cuts 164
10.2 Proof Outline for Our Results 166
10.3 Bounding the Number of k-Cuts for Digraphs in $\mathcal{D}_\alpha$ 168

10.4 Improved Bounds on the Number of k-Cuts for Digraphs in $\mathcal{D}_\alpha$ with Bounded Out-Degeneracy . . . 173

10.5 Sub-Exponential FPT Algorithms for DFAS, Directed Cutwidth and OLA for Digraphs in $\mathcal{D}_\alpha$ . . . 178

10.5.1 Sub-Exponential Algorithm for DFAS . . . 179

10.5.2 Sub-Exponential Algorithm for Directed Cutwidth . . . 181

10.5.3 Sub-Exponential Algorithm for OLA . . . 182

11 Improved Sub-Exponential FPT Algorithms for DFAS and DEOCT on Bounded Independence Number Digraphs 187

11.1 Improved Sub-Exponential FPT Algorithm for DFAS on $\mathcal{D}_\alpha$ . . . 188

11.2 Sub-Exponential FPT Algorithm for DEOCT on $\mathcal{D}_\alpha$ . . . 188

IV Undirected Cut Problems: Exploring and Extending the Tool-Kit 193

12 Balanced Judicious Bipartition is FPT 195

12.1 Some Preliminaries . . . 200

12.2 Solving Balanced Judicious Bipartition . . . 201

12.3 Solving Annotated Bipartite-BJB . . . 204

12.4 Solving Annotated Bipartite Connected-BJB . . . 206

12.5 Solving Favorable Instances of Hypergraph Painting . . . 222

12.5.1 Color Coding The Instance . . . 222

12.5.1.1 Classifying Hyperedges . . . 223

12.5.1.2 Introducing Good Assignments . . . 224

12.5.1.3 Associating the Graph L_p with an Assignment p . . . 226

12.5.1.4 Rules to Modify a Good Assignment . . . 228

12.5.1.5 Constructing a Supergraph L_p^* of L_p . . . 231

12.5.2 Dynamic Programming 233

13 Degeneracy Reduction Preserving Minimal Multicuts 241

13.1 Outline of the Proof 242

13.2 Identifying Vertices Irrelevant to Digraph Pair Cuts 246

13.3 Covering Small Multicuts in a Subgraph Without Highly Connected Set 252

13.4 Finding Large Connected Sets 254

V Vertex Deletion Problems With An Independence Constraint 261

14 Faster Algorithms and Combinatorial Bounds for Independent Feedback Vertex Set 263

14.1 Some Preliminaries 265

14.2 FPT Algorithm for Independent Feedback Vertex Set 266

14.2.1 Algorithm for Disjoint Independent Feedback Vertex Set. 267

14.2.2 A Family of Counter Examples to Song's Algorithm for IFVS 277

14.3 Exact Algorithm for Independent Feedback Vertex Set 277

14.4 On the Number of Minimal Independent Feedback Vertex Sets 281

15 Independent Set Covering Family- Recycling Algorithms for Independent Vertex Deletion Problems 287

15.1 Independence Covering Family and Lemma 290

15.1.1 Extensions 293

15.1.2 Nowhere Dense Graphs 294

15.1.3 Barriers 295

15.2 Algorithms for Bounded Degeneracy Graphs: Trading Independence for Annotations 297

16 Applications of the Independent Set Covering Lemma and Degeneracy Reduction Preserving Minimal Multicuts 303

16.1 Single-Exponential FPT Algorithms for Stable s-t Separator and Stable Odd Cycle Transversal 304

16.2 Stable Multicut is FPT 307

VI Conclusion 309

17 Concluding Remarks and Further Directions 311

Bibliography 316

List of Figures

3.1 (c_1, c_2) is a (u, v) vertex-cut, the green parts correspond to the $\mathcal{Z}(c_i, v)$ and the blue vertices are the vertices of X. P_1 is a path of Type $(u, \square)$, P_2 is a path of Type $(\boxtimes, \boxtimes)$ and P_3 is a path of Type $(\square, \boxminus, v)$ with $y \in Y$. 37

4.1 A directed edge odd cycle transversal (in blue) that is not a directed feedback arc set. 54

7.1 (A) An instance of PVDS. (B) The graph H_v^-. The bold edges correspond to a maximum matching. (C) An application of Rule 7.2.1. . 111

7.2 (A) An instance of PVDS. The gray vertex belongs to HI, and the black vertices belong to HO. (B) An application of Rule 7.2.6. 115

8.1 The structures in the branching rules of PVDS 126

8.2 The structures in the branching rules of OTVDS 136

8.3 The structures in the branching rules of OFVDS 147

10.1 The Vertex Partition for the Sub-exponential XP bound. $\mathcal{P} = \{P_1 \uplus \cdots \uplus P_q\}$ is the vertex partition obtained using chromatic coding and $P_i = P_{i1} \uplus \cdots \uplus P_{i\ell}$ is the partition obtained using Gallai-Milgram's Theorem. Each P_{ij} contains a Directed Hamiltonian Path. The cut arcs of all the cuts that respect $\mathcal{P}$ are marked in blue. 172

10.2 The vertex partition for the Sub-exponential FPT bound. Here the vertices are arranged in the linear order respecting the d-out-degeneracy sequence of D. Here $k = 2$ and the partition of the vertices into the respective sets is demonstrated using appropriate colors. 177

12.1 The construction in the proof of Theorem 12.0.1. 203

13.1 The graphs G, D and D' are displayed in left-to-right order, $T = \{\{s,t\},\{s,t''\},\{s',t'\}\}$ and $T' = \{\{s_1,t_1\},\{s_2,t_1''\},\{s_1',t_1'\}\}$. 248

13.2 Here, the ellipse contains the set of vertices reachable from t in $D'-S$, denoted by R_t. The rectangle colored grey represents $N^+(R_t)$ which includes v 250

13.3 The graph at the right hand side is obtained by the reduction on the graph at the left hand side, where $k = 3$ and Y is the set of black colored vertices. Thick lines represents all possible edges between two sets of vertices. 252

14.1 Reduction Rule 2 268

14.2 Illustration of Reduction Rule 14.2.8. 269

Chapter 17

Concluding Remarks and Further Directions

In Chapters 3 and 4, we presented a sparsification procedure for the class of acyclic digraphs (or more generally, "almost" acyclic) of bounded independence, to preserve, both normal and parity-based, reachability from a given terminal set S to a given terminal set T under the failure of any set of at most k arcs. In particular, it outputs a digraph whose size is completely *independent* of n and polynomial in k, while even the simple classes of directed paths and tournaments admit no sparsifier whose output is a digraph of less than $n-1$ arcs already when $k = 1$. Apart from being interesting on its own from the perspective of fault tolerance, we also showed that our sparsification procedure finds applications in kernelization. Specifically, we proved that the classic Directed Feedback Arc Set problem as well as Directed Edge Odd Cycle Transversal (which, in sharp contract, is W[1]-hard on general digraphs) admit polynomial kernels on bounded independence number digraphs. In fact, for any $p \in \mathbb{N}$, we designed a polynomial kernel for hitting all cycles of length ℓ where ($\ell \mod p = 1$). Additionally, we derived complementary results that assert the NP-hardness of DEOCT on tournaments, as well as its admittance of a subexponential time parameterized algorithm on digraphs of bounded independence. Our result, currently, holds when the input digraph D is "almost acyclic" and has bounded independence number. From the example of the tournament described in the chapter (the one that is obtained by taking a transitive tournament and reversing the arcs along the Hamiltonian path defined by its topological ordering), it seems that some notion of "almost acyclic" might be necessary to have fault tolerant subgraphs whose size avoid the dependence on n. On the other hand, it might be possible to ask for something weaker than bounded independence number. For

example, forbidding the existence of an induced P_α, the directed path on α vertices.

Question 1: Does FTR(S, S) admit a subgraph of size independent of n on digraphs that are "almost acyclic" and have no induced P_α, for some fixed positive integer α?

It is not very difficult to observe that our results (Lemmas 3.1.1 and 4.2.1) also hold when the input graph is undirected and has bounded independence number. It would be interesting (because of the arguments discussed earlier) if one could obtain similar results when the input undirected graph has no induced P_α.

Question 2: Does FTR(S, S) admit a subgraph of size independent of n when the input graph is undirected and has no induced P_α, for some fixed positive integer α?

It would also be interesting to discover other (di)graph classes where the dependence on n of the size of the output subgraph can be sublinear, for example, $\log n$, for FTR(S, S) and also for other fault tolerant graph properties.

Question 3: Apart from Directed Edge Odd Cycle Transversal, which other problems that are W[1]-hard on general digraphs become FPT (or even admit a polynomial kernel) on digraphs of bounded independence?

In Chapter 5, we studied DFVS by parameterizing it by the distance of the input graph to a graph of bounded treewidth, in addition to the solution size.

Question 4: Does DFVS/DFVS+Tw-η Mod admits a uniform kernel? That is, whether there exists a kernel of size $f(\eta)(k\ell)^c$ for DFVS/DFVS+Tw-η Mod, where c is a fixed constant that does not depend on η.

We remark that although our kernel has size $(k\ell)^{\mathcal{O}(\eta^2)}$, the kernelization algorithm runs in time $g(\eta)n^{\mathcal{O}(1)}$ for some function g that depends only on η. Second, another interesting direction of work is to consider the kernelization complexity of DFVS with various other structural parameters. One can also consider the question of designing a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$-time algorithm for DFVS in these settings. In particular, consider the following question.

Question 5: Does DFVS/DFVS+Tw-η Mod admit $2^{\mathcal{O}(k+\ell)}n^{\mathcal{O}(1)}$-time algorithm?

Finally, we reiterate the famous open questions.

Question 6: (i) Does DFVS admit a polynomial kernel, and (ii) does there exist a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithm for DFVS?

In Chapters 6 and 7, we showed that OFVDS admits a kernel of size $\mathcal{O}(k^2)$, and that unless $\mathsf{coNP} \subseteq \mathsf{NP/poly}$, this bound is essentially tight. Furthermore, we

showed that PVDS admits a kernel of size $\mathcal{O}(k^3)$. This result significantly improves upon the one by Mnich and van Leeuwen [164], who showed that PVDS admits a kernel of size $\mathcal{O}(k^{18})$.

Question 7: Does PVDS admit a kernel of size $\mathcal{O}(k^2)$?

In Chapter 10, we designed sub-exponential time parameterized algorithms for DFAS, Directed Cutwidth and OLA on digraphs of bounded independence number. We thus significantly generalized known results for the restricted case of input digraphs that are tournaments. Towards this, we obtained an upper bound on the number of k-cuts in digraphs in $\mathcal{D}_\alpha$. This bound is our main contribution, which we believe will find further implications in the future, and will be of independent interest. On the kernelization front of these problems, we already gave a polynomial kernel for DFAS on digraphs of bounded independence number and a polynomial kernel for OLA already exists for general digraphs [14]. Also, Directed Cutwidth cannot admit a polynomial kernel on semi-complete digraphs but admits a polynomial Turing kernel on semi-complete digraphs [14]. Along these lines an answer to the following question would close the kernelization picture for these problems on the class of bounded independence number digraphs.

Question 8: Does Directed Cutwidth admit a polynomial Turing kernel on $\mathcal{D}_\alpha$?

In Chapter 14, we studied the Independent Feedback Vertex Set problem in the realm of parameterized algorithms, moderately exponential time algorithms and combinatorial upper bounds. We gave a deterministic algorithms for the problem running in times $\mathcal{O}^*(4.1481^k)$ and $\mathcal{O}^*(1.5981^n)$. Finally, we showed that the number of minimal ifvses in any graph on n vertices is upper bounded by 1.7480^n. We also complemented the upper bound result by obtaining a family of graphs where the number of minimal ifvses is at least $3^{n/3}$. We state a few concrete open problems in this context below.

Question 9: Does Independent Feedback Vertex Set admit a kernel of size $\mathcal{O}(k^2)$?

Question 10: Could we close the gap (or even bring it closer) between the upper bound and the lower bounds on the number of minimal ifvses in any graph on n vertices?

In Chapters 13 and 15, we presented two new combinatorial tools for the design of parameterized algorithms. The first was a simple linear time randomized algorithm that given as input a d-degenerate graph G and integer k, outputs an independent

set Y, such that for every independent set X in G of size at most k the probability that X is a subset of Y is at least $\left(\binom{(d+1)k}{k} \cdot k(d+1)\right)^{-1}$. We also introduced the notion of a *k-independence covering family* of a graph G. The second tool was a new (deterministic) polynomial time graph sparsification procedure that given a graph G, a set $T = \{\{s_1, t_1\}, \{s_2, t_2\}, \ldots, \{s_\ell, t_\ell\}\}$ of terminal pairs, and an integer k returns an induced subgraph $G^\star$ of G that maintains *all* of the inclusion minimal multicuts of G of size at most k, and does not contain any $(k+2)$-vertex connected set of size $2^{\mathcal{O}(k)}$. Our new tools yields new FPT algorithms for Stable s-t Separator, Stable Odd Cycle Transversal, and Stable Multicut on general graphs (contained in Chapter 16), and for Stable Directed Feedback Vertex Set on d-degenerate graphs (Chapter 15), resolving two problems left open by Marx et al. [159]. Observe that similar results will hold for a variant of these problems where instead of the solution being independent, one asks for a solution that induces an r-partite graph, for some fixed r. To get this, one can first find a k-independent set covering family and then guess/choose r sets in this family such that each partition of the r-partite solution is contained inside exactly one of the chosen sets. By doing so, we again reduce our problem to an annotated problem where one needs to find a solution which is contained in the union of the r chosen sets. One of the most natural direction to pursue further is to find more applications of our tools than given by us. Apart from this there are several natural questions that arise from our work.

Question 11: In the Stable Multicut problem we ask for a multicut that forms an independent set. Instead of requiring that the solution S is independent, we could require that it induces a graph that belongs to a hereditary graph class $\mathcal{G}$. Thus, corresponding to each hereditary graph class $\mathcal{G}$, we get the problem $\mathcal{G}$-Multicut. Is $\mathcal{G}$-Multicut FPT? Concretely, if $\mathcal{S}$ is the set of forests then is $\mathcal{S}$-Multicut FPT?

Question 12: Given a hereditary graph class $\mathcal{G}$, we can define the notion of k-$\mathcal{G}$ covering family, similar to k-independence covering family. Does there exist other hereditary families, apart from the family of independent sets, such that k-$\mathcal{G}$ covering family of FPT size exists?

Observe that for all the problems whose non-stable version admits a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithm on general graphs, such as s-t Separator and Odd Cycle Transversal, we get a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithm for these problems on graphs of bounded degeneracy. As a corollary, we get a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithm for these problems on planar graphs, graphs excluding some fixed graph H as minor or a topological minor and graphs of bounded degree. A natural question in this regard is the following.

Question 13: Do Stable s-t Separator and Stable Odd Cycle Transversal admit a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithm on general graphs?

Summary

With this thesis we advance the algorithmic tool-kit, and contribute to the existing literature concerning parameterized (di)graph cut problems. The study has been conducted from three broad perspectives. The *first* is the urge to understand and contribute to the otherwise limited algorithmic tool-kit available to tackle digraph (cut) problems. The *second* concerns the exploration of the rich structural and algorithmic tool-kit available to tackle undirected graph cut problems, thereby resulting in engineering some of the existing techniques and exhibiting a broader scope of their applicability, together with contributing more advanced tools to the existing machinery. The *third* concerns building tools that allow "re-usability" of algorithms to solve problems under the presence of certain additional constraints.

Towards the above objectives, the concrete questions that are addressed in the thesis are inspired from some major open problems and concerns in the field of parameterized complexity and kernelization. Some of these being the famously active open problem of the existence of a polynomial kernel for Directed Feedback Vertex/Arc Set, sub-exponential parameterized algorithms for digraph problems beyond tournaments, parameterized algorithms for partitioning problems beyond the classical partitioning problems, the existence of single exponential FPT algorithms for stable versions of classical cut problems and the parameterized complexity of Stable Multicut. We address the above questions either in full or extend the results known in literature that take steps to come closer to resolving the actual question. Below we describe the specific results that we obtain. We also remark that the below mentioned results, in several cases, lead to various enticing insights. In particular, through one of our results, we establish connections between kernelization and fault-tolerant subgraphs. Another result is based on a novel application of important separators in the design of polynomial kernels, which is a rare sight in kernelization. Yet other results give an interesting and useful combinatorial insight about the problem at hand.

The question of the existence of a polynomial kernel for Directed Feedback Vertex/Arc Set (DFV/AS) in general still remain open. We generalize all the results that exist in literature that attempt to come closer to answering this big open question in some way. In particular, the study for the polynomial kernel for DFV/AS has been conducted on three fronts: by restricting the input digraph, by enriching the parameter and by restricting the output DAG. In the first category, positive answers are known for DFAS when the input is a tournament or some variation of it. We extend this result by designing a polynomial kernel for DFAS when the input digraph has a bounded independence number. In the second category, DFVS has been shown to admit a polynomial kernel when the parameter is the feedback vertex set of the underlying undirected graph. We generalize this result by showing that DFVS admits a polynomial kernel when the parameter is the solution size plus the size of a treewidth-η modulator of the underlying undirected graph, for any positive integer η. In the third category, polynomial kernels have been shown for the deletion to out-tree, out-forest and pumpkin problems. We first give improved kernels for the deletion to out-forest and pumpkin problems. We later show that deletion to DAGs of bounded treewidth admit a polynomial kernel.

The second question concerns the non-availability of sub-exponential FPT algorithms for digraph problems beyond tournaments. Towards this, we extend the design of sub-exponential FPT algorithms for various problems on digraphs of bounded independence number.

Towards our study concerning the undirected cut problems, we extend the scope of applicability of the special tree decomposition designed by Cygan et al. [69] to prove that Minimum Bisection is FPT. Bollobás and Scott coined the terms classical and judicious partitioning problems. In classical partitioning problems, the goal is to find a partition that minimizes or maximizes some objective. In judicious partitioning problems, the goal is to find a partition that minimizes or maximizes *several* objectives *simultaneously*. Clearly, going by the definition of these partitioning problems, judicious partitioning problems are harder (in terms of finding algorithms) than classical partitioning problems. We study a judicious partitioning problem called Balanced Judicious Bipartition (BJB) and show, using the special tree decomposition of Cygan et al. [69] together with layers of randomized contractions, that BJB is FPT. Thus, we broaden the scope of applicability of the special tree decomposition of Cygan et al. [69] from designing an FPT algorithm for a classical partitioning problem (Minimum Bisection) to a judicious partitioning problem. Another tool of our study is the treewidth reduction lemma by Marx

et al. [159] that preserves minimal separators of a certain size while bounding the treewidth of the graph as a function of the separator size. The treewidth reduction lemma of Marx et al. [159] also preserves all minimal multicuts of a fixed size but in this case the treewidth of the resulting graph is bounded in terms of both the multicut size and the number of terminal pairs in the input. We design a sparsification procedure that preserves minimal multicuts and does *degeneracy reduction* where the degeneracy is bounded only in terms of the solution size. We later develop another tool that together with the newly designed degeneracy reduction resolves one of the open problems posed by Marx et al. [159].

Finally, we consider vertex deletion problems with an additional constraint of independence on the solution. Towards this, we first design an improved FPT algorithm for Independent Feedback Vertex Set (IFVS) with running time $\mathcal{O}^\star(4.1481^k)$[1] that beat the then best known algorithm for IFVS that ran in time $\mathcal{O}^\star(5^k)$.We then designed a tool that allows to trade the additional constraint of independence with annotations for the design of FPT algorithms for Independent Vertex Deletion problems on bounded degeneracy graphs. Typically, annotations are easy to deal with, compared to the independence constraint, specially for cut problems where a torso of some kind usually work to take care of annotations. With this newly established tool and our tool of degeneracy reduction (from the previous paragraph), and treewidth reduction of Marx et al. [159], we resolve all the open problems posed by Marx et al. in [159], viz. the existence of single-exponential algorithms for Stable s-t Separator and Stable Odd Cycle Transversal, and an FPT algorithm for Stable Multicut.

[1] $\mathcal{O}^\star$ hides polynomial factors.

Chapter 1

About the Thesis

(Di)Graph cut problems constitute a class of problems where given a graph the goal is to find a minimum number of vertices or edges whose deletion makes the resulting graph satisfy some global separation property. A fundamental (di)graph cut problem is the s-t Separator/Cut problem, where given a (di)graph and two fixed vertices s, t of the graph, the goal is to find a minimum number of vertices/edges whose deletion disconnects s from t. It is a classical polynomial time problem whose several natural generalizations turns out to be NP-hard. For example, consider the Multiway Cut (MWC) problem where instead of being given two vertices s, t, one is given a set of vertices $\{t_1, \ldots, t_p\}$ and the goal is to delete a minimum number of edges such that the resulting graph has no path from t_i to t_j, for any $i, j \in \{1, \ldots, p\}$, $i \neq j$. Clearly, when $p = 2$, the MWC is exactly the s-t Cut problem. Another classical cut problem that generalizes both s-t Cut and MWC is the Multicut (MC) problem. In the MC problem one is given a set of pairs of vertices $\{(s_1, t_1), \ldots, (s_p, t_p)\}$ and the goal is to delete a minimum number of edges such that the resulting graph has no path from s_i to t_i, for every $i \in \{1, \ldots, p\}$. Moreover, there are many other graph problems like the Feedback Vertex Set (FVS) and Odd Cycle Transversal (OCT) problems, where the goal is to delete vertices such that the resulting graph has no cycles and odd cycles respectively. This at the first sight may not resemble the traditional cut problem formulations, but it turns out that appropriate combinatorial viewpoints of the problem exhibits that at the heart of these problems lie a cut problem [51, 187]. We will elaborate and explore these relations later in the thesis. Yet another is the Minimum Bisection problem where is goal is to partition the vertex set of the input graph into (roughly) equal parts such that the number of edges with one endpoint in each part is minimized [69].

The broad goal of this thesis is to conduct an algorithmic study of these (di)graph cut problems in the milieu of parameterized complexity and kernelization. Parameterized complexity is the study of multivariate analysis of algorithms for NP-hard problems. Classically the growth of the running time of algorithms is measured as a function of the input size. Typically, the notion of efficiency is associated with algorithms whose running time growth is a polynomial function of the size of the input. For NP-hard problems, unless $P = NP$, there cannot be efficient algorithms. The role of parameterized complexity is to segregate the running time analysis of the algorithms into two separate components – one that measures the growth of the running time as a function of the input size and second that measures the growth as a function of a fixed parameter. Typically, the goal is to design algorithms where the first component described above grows polynomially or show that such algorithms do not exist. Formally speaking, an instance of a parameterized problem is a pair $(\mathcal{I}, k)$, where $\mathcal{I}$ denotes an instance of the problem and k is a parameter. An algorithm for this parameterized problem is said to be *fixed-parameter tractable (FPT)* if its running time is $f(k) \cdot |\mathcal{I}|^{\mathcal{O}(1)}$, where $f()$ is a function that depends on the parameter k alone. If a parameterized problem Π admits such an algorithm then it is said to be FPT with respect to the parameter k. There is also a hardness theory for fixed-parameter tractability. Some problems are unlikely to admit FPT algorithms, under certain complexity theoretic assumptions. For the purposes of this thesis, we call such problems W-hard. FPT algorithms do not just yield efficient algorithms when the input instance has a small parameter but also give an insight as to what makes the problem hard. A field very closely related to parameterized complexity is the field of kernelization. Simply put, kernelization is *efficient pre-processing with guarantees for parameterized problems*. Formally, kernelization of a parameterized problem Π is a polynomial time algorithm that takes an instance $(\mathcal{I}, k)$ of Π and returns an equivalent instance $(\mathcal{I}', k')$ such that $|I'|, k' \leq g(k)$, where $g(\cdot)$ is some function that only depends on the parameter k. In this case, the problem Π is said to admit a kernel of size $g(k)$. If $g(k)$ is a polynomial function, then Π is said to admit a *polynomial kernel*. The connection between FPT and kernelization is made precise with the following theorem that says that a decidable problem is FPT if and only if it admits a polynomial kernel [83]. In fact, if a parameterized problem admits an algorithm with running time $f(k) \cdot |\mathcal{I}|^{\mathcal{O}(1)}$, then it admits a kernel of size $f(k)$.

The typical questions around which the study in this thesis has been conducted concern the existence of FPT algorithms or polynomial kernels for various (di)graph cut problems, or improving the $f(k)$ factor in the running time or the size of the polynomial kernel of the existing algorithms. An important remark to make here is

that the above set of goals were achieved, even though at the forefront, the motives behind the study were more generic. We describe the broad objectives under which the study in this thesis was conducted. These can be classified into 3 categories which are briefed below. We will elaborate on them later.

The first objective was to understand the otherwise mysterious and complex nature of digraphs. A lot of problems on digraphs generally tend to be more difficult to attack than their undirected counterparts. This difficulty is visibly obvious considering the dearth of algorithmic tool-kit for digraph problems. The goal here was to contribute to the otherwise limited understanding of some digraph (cut) problems. This study led to improved and powerful results while establishing new connections between varied fields.

The motivations of the second objective stems from the contrasting rich and tool-kit available for undirected cut problems. The area has received notable attention and has led to the creation and invention of promising techniques. In the second part, we explore some of these promising tools, resulting in engineering some of them to solve harder problems thereby showcasing their wider applicability. We also develop some new tools that finds application in our third part and together with our tools from the next part, resolve all the open problems posed by Marx et al. in [159].

The third objective was to deal with an arguably unlikable trait of algorithms which is their non-reusability under a slight change in the question that they are designed to solve. We design a tool that allows re-usability of algorithms of vertex deletion problems to solve certain variants of it.

The above three objectives are studied in three parts of the thesis. We elaborate on the study towards each of these objectives below.

1.1 Digraph Cut Problems: Advancing the Otherwise Limited Tool-Kit

Even though digraphs model a lot of real life problems so naturally, we are still a long way in understanding the behaviour of problems on general digraphs. Directed variants of most of the cut problems described above viz. Directed MWC (DMWC), Directed MC (DMC), Directed FVS (DFVS) and Directed OCT (DOCT) are exceptionally hard in the sense that the problems are not just NP-hard but also

APX-hard [47, 86, 150, 131, 124, 78]. In the area of parameterized complexity, the problems have been open for a very long time and recently their FPT statuses have been resolved. DMWC and DFVS have been shown to be FPT [57, 51] while the other two are W-hard [178, 150]. On the kernelization complexity front, it is known that DMWC cannot admit a polynomial kernel even when the number of terminals is two [67], whereas the question for DFVS still remains open. Again, on the algorithm design front, the race for asymptotically better running times exist. In particular, the question of the existence of an algorithm for DFV/AS (Directed Feedback Arc Set (DFAS) is the arc deletion version of DFVS) that runs in time $\mathcal{O}^*(2^{\mathcal{O}(k)})$ has been long standing. Yet another looked out algorithms are the ones that run in sub-exponential FPT time, that is, algorithms with running time $\mathcal{O}^*(2^{o(k)})$. Sub-exponentiality in FPT is a rare trait and in the case of digraph problems, it is known only when certain problems are restricted to tournaments.

The first two parts of the thesis is motivated by the open questions concerning the existence of a polynomial kernel for DFV/AS, a $\mathcal{O}^*(2^{\mathcal{O}(k)})$ algorithm for DFVS and the design of sub-exponential FPT algorithms beyond tournaments. We formally define the DFV/AS problem here.

Directed Feedback Vertex/Arc Set (DFV/AS) **Parameter:** k
Input: A digraph D and a non-negative integer k.
Question: Does there exist $S \subseteq V(D)$/ $S \subseteq E(D)$ of size at most k such that $D - S$ is a DAG?

1.1.1 Towards a Polynomial Kernel for DFV/AS

Feedback Set problems form a family of fundamental combinatorial optimization problems. The input for Directed Feedback Vertex Set (DFVS) (Directed Feedback Arc Set (DFAS)) consists of a directed graph (digraph) D and a positive integer k, and the question is whether there exists a subset $S \subseteq V(D)$ ($S \subseteq E(D)$) such that the graph obtained after deleting the vertices (edges) in S is a directed acyclic graph (DAG). Similarly, the input for Undirected Feedback Vertex Set (UFVS) (Undirected Feedback Edge Set (UFES)) consists of an undirected graph G and a positive integer k, and the question is whether there exists a subset $S \subseteq V(G)$ ($S \subseteq E(G)$) such that the graph obtained after deleting the vertices (edges) in S is a forest.

All of these problems, excluding Undirected Feedback Edge Set, are NP-complete. Furthermore, Feedback Set problems are among Karp's 21 NP-complete problems and have been topic of active research from algorithmic [9, 12, 18, 39, 46, 48, 51, 54, 70, 72, 90, 114, 122, 126, 132, 182, 121] as well as structural points of view [88, 123, 127, 180, 186, 192, 193]. In particular, such problems constitute one of the most important topics of research in Parameterized Complexity [39, 48, 51, 54, 70, 72, 126, 122, 132, 182, 121], spearheading development of new techniques.

For over a decade, resolving the parameterized complexity of DFV/AS was considered one of the most important open problems in parameterized complexity (On general digraphs, the problems DFVS annd DFAS are equivalent [65].) . In fact, this question was posed as an open problem in the first few papers on fixed-parameter tractability (FPT) [80, 81]. In a breakthrough paper, DFVS was shown to be fixed-parameter tractable parameterized by the solution size k by Chen et al. [51] in 2008. Specifically, Chen et al. [51] developed an algorithm that solves DFAS in time $\mathcal{O}(k! \cdot 4^k \cdot k^4 \cdot n^{\mathcal{O}(1)})$, based on the powerful machinery of important separators [65]. Subsequently, it was observed that, in fact, the running time of this algorithm is $\mathcal{O}(4^k \cdot k! \cdot k^4 \cdot nm)$ (see, e.g., [65]). Since then, the quest to assert the existence of a polynomial kernel for this problem has been unfruitful. Over the years, it has been repeatedly posed as a major challenge in the subfield of Kernelization [65, 83, 158, 155] (also see [1] for a number of workshops and schools where it was posed as an open problem). In fact, the two specific problems whose polynomial kernelization complexity is completely unknown and their resolution is raised most frequently are DFAS and Multiway Cut [65, 83]. The existence of a polynomial kernel for DFVS is open even when the input digraph is a planar digraph. At the front of parameterized algorithms, the recent work by Lokshtanov et al. [149] improved upon the polynomial factor of the aforementioned algorithm by the design of an $\mathcal{O}(k!4^k k^5(m+n))$-time algorithm. It is known that unless the Exponential Time Hypothesis (ETH) is false, parameterized by the treewidth tw of the underlying undirected graph, DFAS cannot be solved in time $2^{o(\mathsf{tw}\log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$ [33]. However, it is unknown whether DFAS is solvable in time $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$. In this regard, the only lower bound known is of $2^{\Omega(k)} \cdot n^{\mathcal{O}(1)}$ under the ETH [65, 149].

Particular attention has been given to the parameterized complexity of DFAS on tournaments. The classical complexity (NP-hardness) of DFAS on tournaments has a curious history. More than two decades ago, this problem was conjectured to be NP-hard by Bang-Jensen and Thomassen [11]. In 2008, Ailon et al. [4] proved that

this problem does not admit a polynomial-time algorithm unless NP⊆BPP. Later, the reduction of Ailon et al. [4] was derandomized independently by Alon [5] and Charibt et al. [41], to prove that DFAS on tournaments is NP-hard. With respect to Parameterized Complexity, Alon et al. [7] proved that DFAS on tournaments admits a sub-exponential time parameterized algorithm. We talk about this set of literature in more detail in the next section. On tournaments, Bessy et al. [20] have proved that DFAS admits a linear-vertex kernel (improving upon polynomial kernels given in [7, 79]).

As of now, the question regarding the polynomial kernel for DFV/AS has been attacked at from the following different viewpoints. Various restrictions have been considered like restricting the input of the problem, enriching the parameter or restricting the output. We explore and extend the results obtained in all these different directions.

1.1.1.1 DFAS: Restricting the Input

Polynomial kernels have been known for DFAS when the input digraph is restricted to be a tournament or certain variation of a tournament like a semi-complete digraph, a bipartite tournament [163], a locally tournament [10] or some Φ-decomposable digraphs including quasi-transitive digraphs [10]. In this thesis, we extend the polynomial kernel result for DFAS to the class of digraphs of bounded independence number. This class of digraphs was first considered by Fradkin and Seymour to extend the polynomial time algorithm result for the Edge Disjoint Paths problem for fixed number of paths, beyond tournaments [106]. Our result is obtained by establishing connections of kernelization with *fault-tolerant subgraphs*. In particular, to design a polynomial kernel for DFAS on the class of bounded independence number digraphs, we establish its connection with *reachability preserving fault-tolerant subgraphs* and prove results for reachability preserving fault-tolerant subgraphs. We take this connection further by proving similar results for *parity reachability fault-tolerant subgraphs* thereby leading to a polynomial kernel for Directed Edge Odd Cycle Transversal (DEOCT) problem (defined below) when the input digraph has bounded independence number.

Directed Edge Odd Cycle Transversal (DEOCT) **Parameter:** k
Input: A digraph D and a non-negative integer k.
Question: Does there exist $S \subseteq E(D)$ of size at most k such that $D - S$ has no directed odd cycles?

This is the first polynomial kernel that has been designed for the DEOCT problem. Also these connections have extended their contributions to the field of fault-tolerant subgraphs too. The kernelization result of DEOCT is complemented by showing that DEOCT is NP-hard on tournaments. We obtain this result by proving that DEOCT on tournaments is equivalent to DFAS on tournaments. This set of results appeared in the proceedings of the *11th Innovations in Theoretical Computer Science (ITCS) 2020*.

In addition to the rich history of the DFAS problem in theoretical studies, the elimination of directed feedback loops is highly relevant to rank aggregation, Voting Theory, the resolution of inconsistencies in databases, and the prevention of deadlocks [195, 20, 110, 130, 51, 92]. While in a wide-variety of applications, *most* relations between the entities in a network are both present and known, it is generally unrealistic (in real-world partial and noisy data) that *all* relations will be so. Then, the usage of a bounded independence digraphs naturally comes into play.

1.1.1.2 DFVS: Enriching the Parameter

Bergougnoux et al. [19] studied the DFVS problem with a possibly bigger parameter (bigger than the solution size), the size of the *feedback vertex set (fvs)* of the underlying undirected graph. They designed a polynomial kernel for DFVS parameterized by fvs. In this thesis, we design a polynomial kernel for DFVS parameterized by the solution size plus the size of a treewidth-η modulator of the underlying undirected graph, for any positive integer η. A treewidth-η modulator of an undirected graph is a set of vertices whose deletion results in a graph of treewidth at most η. Observe that fvs is a treewidth-1 modulator. Since parameterized by the size of a treewidth-η modulator alone, for any $\eta \geq 2$, DFVS cannot have a polynomial kernel (under reasonable complexity assumptions) [68], our polynomial kernel result for DFVS generalizes that of Bergougnoux et al. [19]. The main highlight of our result is a polynomial time procedure that computes the union of solutions of exponentially many instances of Skew Multicut that have small solutions. We believe that this procedure could be of independent interest when considering the design of a "real" polynomial kernel for DFVS. This result appeared as a brief announcement in the proceedings of the *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)* and in the proceedings of the *Algorithms and Data Structures Symposium (WADS 2019)*.

1.1.1.3 DFVS: Restricting the Output

Mnich and Leewan [164] started the study of problems of deletion to classes of acyclic digraphs. In particular, they considered the classes of out-forests, out-trees and (directed) pumpkins. An *out-tree* is a digraph where each vertex has in-degree at most 1 and the underlying (undirected) graph is a tree. An *out-forest* is a disjoint union of out-trees. A digraph is a *pumpkin* if it consists of a source vertex s and a sink vertex t, $s \neq t$, together with a collection of internally vertex-disjoint induced directed paths from s to t. Here, all vertices except s and t have in-degree 1 and out-degree 1. The corresponding deletion problems are defined below.

Out-Forest Vertex Deletion Set (OFVDS) **Parameter:** k
Input: A digraph D and a positive integer k.
Question: Is there a set $S \subseteq V(D)$ of size at most k such that $F = D - S$ is an out-forest?

Out-Tree Vertex Deletion Set (OTVDS) and Pumpkin Vertex Deletion Set (PVDS) are defined in a similar manner, where instead of an out-forest, F should be an out-tree or a pumpkin, respectively. Mnich and van Leeuwen [164] showed that OFVDS and OTVDS admit kernels of size $\mathcal{O}(k^3)$ and PVDS admits a kernel of size $\mathcal{O}(k^{18})$. The intention of their project was that restricting the output digraph to be a special kind of a DAG could bring us closer to understanding the DFVS (deletion to DAGs) problem. We initially improved their kernel results by designing kernels of size $\mathcal{O}(k^2)$ and $\mathcal{O}(k^3)$ for Out-forest and Pumpkin deletion respectively. We also prove that the size $\mathcal{O}(k^2)$ is asymptotically tight under reasonable complexity assumptions. These results appeared at the *Journal of Computer and System Sciences (JCSS 2017)* and in the proceedings of the *27th International Symposium on Algorithms and Computation (ISAAC 2016)*.

For the mentioned problems, we also designed faster FPT algorithms with running times $\mathcal{O}^*(2.732^k)$, $\mathcal{O}^*(2.562^k)$ and $\mathcal{O}^*(2.562^k)$ for Out-forest, Out-tree and Pumpkin deletion problems respectively. These results appeared at the *Theory of Computing Systems (TOCS 2018)*.

We later designed an algorithm using the existing tools from the protrusion machinery and showed that in fact, the problem of deletion to DAGs of bounded treewidth ($\mathcal{F}_\eta$-Vertex Deletion, defined below) admits a polynomial kernel. Here, by treewidth of a directed graph we mean the treewidth of its underlying

undirected graph. Observe that all out-forest, out-tree and pumpkin have treewidth at most 2. This result appeared in the proceedings of the *Algorithms and Data Structures Symposium (WADS 2019)*. For a positive integer $\eta > 0$, let $\mathcal{F}_\eta$ denote the family of digraphs of treewidth at most η.

$\mathcal{F}_\eta$-Vertex Deletion	**Parameter:** k
Input: A digraph D and a non-negative integer k.	
Question: Does there exist a set of at most k vertices, say S, such that $D - S \in \mathcal{F}_\eta$?	

1.1.2 Sub-Exponential FPT Algorithms Beyond Tournaments

The next question we address concerns sub-exponential FPT algorithms. These are the algorithms whose running times are of the form $2^{o(k)} \cdot n^{\mathcal{O}(1)}$, where k is the parameter and n is the input size. Sub-exponential FPT algorithms for problems on digraphs had so far only been known for certain problems when the input digraph is a tournament. We extend this barrier and designed sub-exponential FPT algorithms for certain digraph problems (including DFAS and DEOCT) when the input digraph has bounded independence number. Further motivations of this study are elaborated below.

Tournaments form one of the most well studied families of digraphs, both algorithmically and structurally. In particular, whenever we try to generalize results that hold for undirected graphs to digraphs, arguably, one of the first families to consider is that of tournaments. Indeed, this has been the case when designing parameterized algorithms or approximation algorithms. Two problems that have been extensively studied on tournaments are Directed Feedback Vertex Set (DFVS) and Directed Feedback Arc Set (DFAS).

In the realm of approximation, we know that DFVS admits a 7/2-approximation algorithm on tournaments [165]. Recently it has been shown that DFVS admits a 2-approximation via a randomized algorithm and a 2-approximation in deterministic quasi-polynomial time [142]. On the other hand, DFAS admits a PTAS on tournaments [130]. For DFVS on tournaments, the best known parameterized algorithm runs in time $1.618^k \cdot n^{\mathcal{O}(1)}$ [137]. Prior to this the fastest known parameterized algorithm for DFVS ran in time $2^k \cdot n^{\mathcal{O}(1)}$ [79], based on iterative compression. As in the case of approximation, from the viewpoint of Parameterized Complexity, DFAS

on tournaments is "easier" than DFVS on tournaments. Here, we mean that for DFAS on tournaments, sub-exponential time parameterized algorithms are known. The quest for sub-exponential time parameterized algorithms for DFAS has a rich history. For a long time (even after the $2^k \cdot n^{\mathcal{O}(1)}$-time algorithm for DFVS was discovered), the question of the existence of an algorithm for DFAS that runs in time $2^k \cdot n^{\mathcal{O}(1)}$ was still being posed as an open problem. Based on a generic method called chromatic coding, Alon et al. [7] gave the first sub-exponential time parameterized algorithm for DFAS, which runs in time $2^{\mathcal{O}(\sqrt{k}\log^2 k)} \cdot n^{\mathcal{O}(1)}$. This was the the first problem not confined to planar graphs (or generalizations such as apex-minor-free graphs) that was shown to admit a sub-exponential time parameterized algorithm. Later, simultaneously and independently, Feige [91] and, Karpinski and Schudy [125] gave faster algorithms that run in time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$. Fomin and Pilipczuk [102] presented a general approach, based on a bound on the number of k-cuts (defined below) in transitive tournaments, that achieved the same running time for DFAS. Using this framework they also designed the first sub-exponential time algorithms for Directed Cutwidth and Optimal Linear Arrangement (OLA) (defined below) on semi-complete digraphs.

Towards the definition of Directed Cutwidth, let D be a digraph. For $X, Y \subseteq V(D)$, let $E(X,Y) = \{(u,v) \in E(D) \mid u \in X, v \in Y\}$ denote the set of arcs from X to Y. For an integer q, denote $[q] = \{1, \ldots, q\}$. The *width* of an ordering $(v_1, \ldots, v_n)$ of $V(D)$ is $\max_{i \in [n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})|$. The *cutwidth* of D, denoted by $\mathbf{ctw}(D)$, is the smallest possible width of an ordering of $V(D)$. Now, Directed Cutwidth is defined as follows.

Directed Cutwidth	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Is $\mathbf{ctw}(D) \leq k$?	

Towards the definition of OLA, let D be a digraph. For two integers i, j, let $[i > j]$ evaluate to 1 if $i > j$, and to 0 otherwise. The *cost* of an ordering $\sigma = (v_1, \ldots, v_n)$ of $V(D)$ is $\sum_{(v_i,v_j) \in E(D)} (i-j) \cdot [i > j]$. In other words, every arc (v_i, v_j) directed backward in σ costs a value equal to its length, where the length of (v_i, v_j) is the distance between v_i and v_j in σ. Our last problem seeks an ordering of cost at most k.

Optimal Linear Arrangement (OLA)	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Is there an ordering of $V(D)$ of cost at most k?	

Barbero et al. [14] studied Directed Cutwidth and OLA on semi-complete digraphs (that is, digraphs where for any two vertices u and v, at least one of the arcs (u, v) and (v, u) is present) and showed that these problems are NP-complete on semi-complete digraphs. Furthermore, they showed that Directed Cutwidth does not a admit polynomial kernel on semi-complete digraphs but admits a polynomial Turing kernel. Finally, they obtained a linear vertex kernel for OLA on general digraphs.

The measure of directed cutwidth plays a key role in the work of Chudnovsky and Seymour [60] where it is shown that tournaments are well-quasi-ordered under immersion. This measure was considered by Chudnovsky et al. [59] also in their algorithmic study of Immersion on tournaments. Later, Fradkin and Seymour [105] showed that the Directed Pathwidth and Topological Containment problems on tournaments are fixed parameter tractable (FPT). Fomin and Pilipczuk [101, 102], and Pilipczuk [179] revisited these problems and gave the best known algorithms for them on tournaments. Fradkin and Seymour [107], in order to generalize their results from tournaments to broader families of graphs, introduced the idea of digraphs that have bounded independence number. In particular, tournaments have independence number 1. They showed that Edge disjoint Paths admits an XP algorithm (that is, an algorithm with running time of the form of $n^{f(k)}$, where n is the number of vertices in the input graph and k is the number of pairs between which one is asked to find edge-disjoint paths) on this family of graphs.

We study well-known cut problems (DFAS, Directed Cutwidth and OLA) on digraphs of bounded independence number. Our main contribution is proving a sub-exponential FPT bound on the number of k-cuts (defined below) in the Yes instances of these problems, which shows that the sub-exponential behaviour of these problems on tournaments generalizes to digraphs of bounded independence number. The running times of all these algorithms have a dependence of the size of the largest input set in the input digraph, on the exponent of n. These results appeared in the proceedings of the *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*.

Later, using our kernelization results of Section 1.1.1.1, we give a sub-exponential FPT algorithm for DEOCT on digraphs of bounded independence number. We also design a new sub-exponential FPT algorithm for DFAS on digraphs of bounded independence number. The running times of both these algorithms do not have any dependence of the size of the maximum independent set in the input digraph, on the exponent of n (where n is the number of vertices in the input digraph). This result for DFAS clearly improves our previous result in terms of the asymptotic running time. These results appeared in the proceedings of the *11th Innovations in Theoretical Computer Science (ITCS) 2020*.

1.2 Undirected Cut Problems: Exploring and Extending the Rich Tool-Kit

In contrast to digraphs, the undirected cut problems have a rich and promising algorithmic tool-kit. We focus on the tools and techniques developed for parameterized algorithms for undirected cut problems over the last decade. We mainly focus on three of them: the randomized contractions of Chitnis et al. [164], the special tree decomposition with highly connected bags of Cygan et al. [69] and the treewidth reduction preserving minimal separators of Marx et al. [159]. With respect to the special tree decomposition of Cygan et al., the tree decomposition was designed to resolve the FPT status of a classical partitioning problem, the Minimum Bisection problem [69]. Applications of this tree decomposition were not known beyond this problem until our work. We extend the utility of this tree decomposition by using it, with layers of randomized contractions, to prove that Balanced Judicious Bipartition (BJB) is FPT.

Balanced Judicious Bipartition (BJB)	**Parameter:** $k_1 + k_2$
Input: A multi-graph G, and integers μ, k_1 and k_2	
Question: Does there exist a partition (V_1, V_2) of $V(G)$ such that $\lvert V_1 - V_2\rvert \leq 1$ and for all $i \in \{1, 2\}$, it holds that $\lvert E(G[V_i])\rvert \leq k_i$?	

BJB is a judicious partitioning problem, a term coined by Bollobas and Scott, [25], where unlike classical partitioning problems, the goal is to find a partition of the graph which minimizes/maximizes over *several* sets of constraints *simultaneously*. This criteria of minimizing/maximizing over several sets together makes judicious

partitioning problems harder compared to the classical partitioning problems. Thus, we broaden the scope of applicability of the special tree decomposition of [69]. This result appeared at *SIAM Journal of Discrete Mathematics (SIDMA 2019)* and in the proceedings of the *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017)*.

The treewidth reduction of Marx et al. [159] is an algorithm that preserves all minimal s-t separators of size at most k of the given graph in a graph of treewidth at most $2^{\mathcal{O}(k)}$. We design a tool that, given a graph G and a set of pairs of vertices $T = \{(s_1, t_1), \ldots, (s_p, t_p)\}$ preserves all minimal T-multicuts of G of size at most k in an induced subgraph of degeneracy at most $2^{\mathcal{O}(k)}$. A T-multicut of G is a set of vertices whose deletion results in a graph that has no path from s_i to t_i for any $i \in \{1, \ldots, \ell\}$. We call this *Degeneracy Reduction Preserving Multicuts*. This tool finds useful applications in the last part of the thesis where we resolve all the open problems posed by Marx et al. in [159]. This result appeared at the *Transactions on Algorithms 2020 (TALG 2020)* and in the proceedings of the *ACM-SIAM Symposium on Discrete Algorithms 2018 (SODA 2018)*.

1.3 Towards the Re-Usability of Algorithms

In this section, we study vertex deletion problems where the solution additionally has a constraint that it is required to form an independent set. Towards this, we first study the Independent Feedback Vertex Set (IFVS) problem.

Independent Feedback Vertex Set (IFVS) **Parameter:** k
Input: An undirected graph G and a positive integer k.
Question: Is there a set S of size at most k such that $G - S$ has no cycles and S is an independent set in G?

We gave a $\mathcal{O}^\star(4.1481^k)$ algorithm for IFVS beating the then best algorithm that ran in time $\mathcal{O}^\star(5^k)$ (the current best known algorithm for this problem runs in time $\mathcal{O}^\star((1+\phi^2)^k)$ where $\phi < 1.619$ is the golden ratio). This result was also complemented by other results for IFVS viz. an exact algorithm with running time $\mathcal{O}^\star((1.5981)^n)$ and an upper bound of 1.7485^n on the number of minimal independent feedback vertex sets in a graph of n vertices. These result appeared in the proceedings of the *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*.

Our parameterized algorithm (and also the existing parameterized algorithm) for IFVS had to be designed from scratch and the additional constraint of independence on the solution could not be tackled while re-using the existing algorithms for FVS. This concern of not being able to use existing algorithms to solve the same problem with some additional constraint (in this case the constraint being that the solution is also required to form an independent set) led to the development of our next tool. We call it the *Independence Set Covering Lemma (ISCL)*. Using this tool, one can re-use algorithms of vertex deletion problems, especially cut problems, for designing algorithms for their variant with the independence constraint. Using our ISCL, our degeneracy reduction preserving minimal multicuts and Marx et al. [159] treewidth reduction preserving minimal separators, we resolve all the open problems posed by Marx et al. [159], viz. the design of single exponential FPT algorithms for Stable s-t Separator (defined below) and Stable Odd Cycle Transversal (defined below), and also the design of an (single-exponential) FPT algorithm for Stable Multicut (defined below). These results appeared in the proceedings of the *ACM-SIAM Symposium on Discrete Algorithms 2018 (SODA 2018)*.

Stable s-t Separator	**Parameter:** k
Input: An undirected graph G, vertices s, t of G and a positive integer k.	
Question: Is there a set S of at most k vertices of G such that $G - S$ has no path from s to t and S is an independent set in G?	

Stable Odd Cycle Transversal	**Parameter:** k
Input: An undirected graph G and a positive integer k.	
Question: Is there a set S of at most k vertices of G such that $G - S$ has odd cycle, that is, $G - S$ is bipartite and S is an independent set?	

Stable Multicut	**Parameter:** k
Input: An undirected graph G, a set $T = \{(s_1, t_1), \ldots, (s_\ell, t_\ell)\}$ of pairs of vertices of G and a positive integer k.	
Question: Is there a set S of at most k vertices of G such that $G - S$ has no path from s_i to t_i for any $i \in \{1, \ldots, \ell\}$ and S is an independent set?	

1.4 Organization of the Thesis

Part II contains the results discussed in Section 1.1.1. Part III contains the results discussed in Section 1.1.2. Part IV contains the results discussed in Section 1.2. Part V contains the results discussed in Section 1.3. Finally, Part VI winds up the thesis with concluding remarks.

Chapter 2

Preliminaries

2.1 Parameterized Complexity and Kernelization

We begin by defining the notion of a parameterized problem.

Definition 2.1.1 (Parameterized Problem)**.** *A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed and finite alphabet. Any $(\mathcal{I}, k) \in \Sigma^* \times \mathbb{N}$ is called an instance of L. $(\mathcal{I}, k)$ is called a* YES *instance of L if $(\mathcal{I}, k) \in L$, otherwise it is called a* NO *instance. For an instance $(\mathcal{I}, k)$, k is called the* parameter*.*

We now define fixed-parameter tractability.

Definition 2.1.2 (Fixed-Parameter Tractability)**.** *Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that L is fixed-parameter tractable (FPT) if there exists a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant c and an algorithm $\mathcal{A}$ that takes as input an instance $(\mathcal{I}, k)$ of L, runs in time $f(k) \cdot |\mathcal{I}|^c$ and correctly decides if $(\mathcal{I}, k) \in L$.*

We next define kernelization.

Definition 2.1.3 (Kernelization)**.** *Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that L admits a kernelization algorithm or kernel, if there exists a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant c and an algorithm $\mathcal{A}$ that takes as input an instance $(\mathcal{I}, k)$ of L and outputs another instance $(\mathcal{I}', k')$ in time $|\mathcal{I}|^c$ with the following properties.*

- *$(\mathcal{I}, k) \in L$ if and only if $(\mathcal{I}', k') \in L$, and*

- $|\mathcal{I}'| + k' = f(k)$.

If $f(k)$ is a polynomial function in k, we say that L admits a polynomial kernel.

While designing our kernelization algorithm, we might be able to determine whether the input instance is a YES instance or a NO instance. For the sake of clarity, in the first case, we simply return YES, and in second case, we simply return NO. To properly comply with the definition of a kernel, the return of YES and NO should be interpreted as the return of a trivial YES instance and a trivial NO instance, respectively.

To design our kernelization algorithm, we rely on the notion of a *reduction rule*. A reduction rule is a polynomial-time procedure that replaces an instance $(\mathcal{I}, k)$ of a parameterized problem L by another instance $(\mathcal{I}', k')$ of L. Roughly speaking, a reduction rule is useful when the instance $\mathcal{I}'$ is in some sense "simpler" than the instance $\mathcal{I}$. In particular, it is desirable to ensure that $k' \leq k$. If a parameterized problems admits a kernelization algorithms that comprise of only useful (as defined above) reduction rules, then it is said to admit a *proper kernel*. The rule is said to be *safe* if $(\mathcal{I}, k)$ is a YES instance of L if and only if $(\mathcal{I}', k')$ is a YES instance of L.

Today parameterized complexity is an established and dynamically developing area of algorithms and various monographs and textbooks are available on this topic. More familiarity can be gained in this field by referring to the books of Downey and Fellows [82], Flum and Grohe [93], Niedermeier [173], and the more recent books by Cygan et al. [65] and Fomin et al. [100].

2.2 Notations and Basic Definitions

To describe the running times of our algorithms, we will use the $\mathcal{O}^\star$ notation. Given $f : \mathbb{N} \to \mathbb{N}$, we define $\mathcal{O}^\star(f(n))$ to be $\mathcal{O}(f(n) \cdot p(n))$, where $p(\cdot)$ is some polynomial function. That is, the $\mathcal{O}^\star$ notation suppresses polynomial factors in the running-time expression.

General Notation. We use $\mathbb{N}$ to denote the set of natural numbers starting from 0. For $i, j \in \mathbb{N}$, $[i], [i]_0, [i, j]$ are shorthands for $\{1, \ldots, i\}$, $\{0, 1, \ldots, i\}$ and $\{i, i+1, \ldots, j\}$ respectively. For a set S, S^2 denotes the set of ordered pairs of S, that is $S^2 = \{(u, v) \mid u \in S, v \in S\}$. For two sets A, B, $A \uplus B$ denotes the disjoint union of A and B. For a set A and $t \in \mathbb{N}$, we use 2^A and $\binom{A}{t}$ to denote the

power set of A and the set of subsets of A of size t, respectively. For a partition $\mathcal{P} = P_1 \uplus \cdots \uplus P_\ell$, each P_i is referred to as a *part* of $\mathcal{P}$. Let $f : A \to B$ be some function. Given $X \subseteq A$, the notation $f(X) = b$ indicates that for all $a \in X$, it holds that $f(a) = b$. The *restriction* $f|_X$ of f is a function from X to B such that for any $a \in X$, $f|_X(a) = f(a)$. An *extension* f' of the function f is a function whose domain, Y, is a superset of A and whose range is B, such that for all $a \in A$, it holds that $f'(a) = f(a)$. For a function $f : D \to R$, $X \subseteq D$ and $Y \subseteq R$, we denote $f(X) = \{f(x) : x \in X\}$ and $f^{-1}(Y) = \{d : f(d) \in Y\}$. The following fact follows from Stirling's approximation.

Fact 2.2.1 ([62])**.** *For all positive integers n, k, $k \leq n$,*

$$\frac{1}{n}\left[\left(\frac{k}{n}\right)^{-k}\left(\frac{n-k}{n}\right)^{-(n-k)}\right] \leq \binom{n}{k} \leq \left[\left(\frac{k}{n}\right)^{-k}\left(\frac{n-k}{n}\right)^{-(n-k)}\right]$$

.

(Di)graphs: For a (di)graph D, $V(D)$ denotes the vertex set of D and $E(D)$ denotes the edge set of D. An edge of D is represented by either (u, v) or uv. If (u, v) (or uv) is an edge of a digraph then u is the tail of this edge and v is the head of this edge. For any $X \subseteq V(D)$ (resp. $X \subseteq E(D)$), $D - X$ denotes the (di)graph obtained by deleting the vertices (resp. edges) of X and $D[X]$ is the subgraph of D induced by X. We sometimes denote $D - X$ by $D \setminus X$ too. When $X = \{v\}$, we use $D - v$ to denote the graph $D - \{v\}$. For a graph G and an edge $e \in E(G)$, G/e denotes the graph obtained after contracting e in G. By $\overleftarrow{D}$, we denote the digraph obtained from D by reversing each of its arcs. If D is an undirected graph, for any $v \in V(D)$, $N_D(v)$ denotes the neighbors of v in D, that is, $N_D(v) = \{u : uv \in E(D)\}$. For a subset $X \subseteq V(G)$, $N(X) = \cup_{v \in X} N(v) \setminus X$. The degree of a vertex v in D, denoted by $deg_D(v)$ or $d_D(v)$, is equal to the number of neighbors of v in D, that is, $deg_D(v) = |N_D(v)|$. The minimum degree of D is the minimum over the degrees of all its vertices. For $X \subseteq V(D)$, we denote by $\delta_D(X)$ the set of boundary vertices of X in D, that is, $\delta_D(X) = \{v \in X :$ there exists $u \in V(G) \setminus X$ such that $\{u, v\} \in E(G)\}$. Whenever the graph D is clear from the context, we drop the subscript D in $N_D(v), d_D(v), deg_D(v), \delta_D(X)$. For any positive integers a, b, we denote by $K_{a,b}$ the (undirected) complete bipartite graph with a vertices in one part and b vertices in the other part.

If D is a digraph, then for any $v \in V(D)$, $N_D^+(v)$ (resp. $N_D^-(v)$) denotes the set of out-neighbours (resp.in-neighbours) of v in D, that is $N_D^+(v) = \{u \in V(D) \mid (v, u) \in E(D)\}$ (resp. $N_D^-(v) = \{u \in V(D) \mid (u, v) \in E(D)\}$). We denote the

in-degree of a vertex v by $d_D^-(v) = |N^-(v)|$ and its out-degree by $d_D^+(v) = |N^+(v)|$. Whenever the (di)graph D is clear from the context, we drop the subscript D in $N_D^+(v), N_D^-(v), d_D^-(v), d_D^+(v), deg_D(v), d_D(v)$. For a set $X \subseteq V(D)$, we let $N_X^-(v)$ (resp. $N_X^+(v)$) denote the set of in-neighbors (resp. out-neighbors) of v in X, that is, $N_X^-(v) = N_D^-(v) \cap X$ (respectively, $N_X^+(v) = N_D^+(v) \cap X$). For any $X, Y \subseteq V(D)$, $E(X, Y)$ denotes the set of arcs of D with tail in X and head in Y, that is, $E(X, Y) = \{(u, v) \in E(D) \mid u \in X, v \in Y\}$. By $d_D(X, Y)$ we denote the number of edges from X to Y in D, that is, $d_D(X, Y) = |E_D(X, Y)|$. When $X = \{u\}$ and $Y = \{v\}$ are singleton sets, we denote $d_D\{u\}, \{v\}$ by $d_D(u, v)$. A digraph D is called *strongly connected* if for each $u, v \in V(D)$ there is a path from u to v and, a path from v to u in D. A set $X \subseteq V(D)$ is called a *strongly connected component* of D if $D[X]$ is a strongly connected digraph and for each $X' \supseteq X$, $D[X']$ is not a strongly connected digraph. A *tournament* is a digraph where there is exactly one arc between each pair of vertices. A digraph with no cycles is called a *directed acyclic graph (DAG)*. A tournament with no cycles is called a *transitive tournament*. For a directed graph D, by the *underlying undirected graph* of D we refer to the simple, undirected graph with vertex set $V(D)$ and arc set $E(D)$. For a directed graph D, by the *(weakly) connected components* of D, we refer to the connected components of the underlying undirected graph of D. Let C be an induced subgraph of D. A set $X \subseteq V(D)$ is called an *independent set* of D if for any $u, v \in X$, $(u, v) \notin E(D)$ and $(v, u) \notin E(D)$. In other words, X is an independent set in the underlying undirected graph of D. The independence number of a digraph is equal to the size of the maximum independent set it contains.

Paths: A *path* P is a (di)graph such that there exists an ordering $(v_1, \ldots, v_q)$ of its vertex set $V(P)$ such that $E(P) = \{(v_i, v_{i+1}) \mid i \in [q-1]\}$. Such a path is called a (v_1, v_q)*-path*, v_1, v_q are called the *end-points of* P and $v_2, \ldots, v_{q-1}$ are called the *internal vertices of* P. A path P is *even* (resp. odd) if the number of arcs/edges in it is even (resp. odd). For any $X, Y \subseteq V(D)$, a path from X to Y refers to a path from some vertex in X to some vertex in Y. We say that P is a path in the digraph D if P is a subgraph of D. We say that P is an *induced path* in D if P is an induced subgraph of D. For paths P and P', by $P \circ P'$ we denote the composition of P and P', that is, the path obtained by appending P' after P. For paths $P, P_1, P_2, \ldots, P_q$ such that $P = P_1 \circ P_2 \circ \ldots \circ P_q$, we say that $P_1 \circ P_2 \circ \ldots \circ P_q$ is a *partition* of P. Paths $\{P_1, \ldots, P_q\}$ are *internally vertex-disjoint* if for all distinct $i, j \in [q]$, the sets of internal vertices of P_i and P_j are disjoint. A *directed Hamiltonian path* in D is a directed simple path on all vertices in D.

2.3 Degeneracy of Graphs

For a non-negative integer d, a graph G is called a *d-degenerate graph* if for every subgraph H of G there exists $v \in V(H)$ such that $deg_H(v) \leq d$. The degeneracy of a graph G, denoted by $\mathsf{degeneracy}(G)$, is the least integer d, for which G is d-degenerate. If for each subgraph H of G, the minimum degree of H is at least d, we say that the degeneracy of G is at least d. For a d-degenerate graph G, a *d-degeneracy sequence* of G is an ordering of the vertices of G, say $\sigma : V(G) \to [|V(G)|]$, such that σ is a bijection and, for any $v \in V(G)$, $|N_G(v) \cap \{u : \sigma(u) > \sigma(v)\}| \leq d$. For a given degeneracy sequence σ and a vertex $v \in V(G)$, the vertices in $N_G(v) \cap \{u : \sigma(u) > \sigma(v)\}$ are called the *forward neighbors* of v in σ, and this set of forward neighbors is denoted by $N^f_{G,\sigma}(v)$. The following proposition says we can find d-degeneracy sequence of a graph in linear time.

Proposition 2.3.1 ([161])**.** *If G is a d-degenerate graph, for some non-negative integer d, then a d-degeneracy sequence of G exists and can be found in time $\mathcal{O}(n + m)$.*

2.4 Graph (Important) Separators

For a (di)graph G, $X, Y \subseteq V(G)$, an *X-Y-separator* in G is a subset $C \subseteq V(G)$, such that there is no path from a vertex in $X \setminus C$ to a vertex in $Y \setminus C$ in $G - C$. For $s, t \in V(G)$ an *s-t-separator* in G is a subset $C \subseteq V(G) \setminus \{s, t\}$ such that there is no path from s to t in $G - C$. The size of a separator is equal to the cardinality of the separator. A *minimum s-t-separator* in G is the one with the minimum number of vertices. A set $Y \subseteq V(G)$ is a *mincut* of G if Y is a smallest set of vertices such that $G - Y$ has at least two components.

Since, checking whether there is an s-t-separator of weight at most k (here a non-negative integer weight function on $V(G)$ is given) can be done by running at most k rounds of the classical Ford-Fullkerson algorithm, Proposition 2.4.1 follows.

Proposition 2.4.1. *Given a (di)graph G, $s, t \in V(G)$, an integer k and $w : V(G) \to \mathbb{N}$, an s-t-separator of weight at most k, if it exists, can be found in time $\mathcal{O}(k \cdot (n + m))$. Also, a minimum s-t-separator can be found in time $\mathcal{O}(mn)$.*

The following proposition follows directly from the standard reduction that reduces finding minimum vertex separators to finding minimum edge cuts in directed

graphs and the result about the later in [118].

Proposition 2.4.2 ([118])**.** *A mincut of a (di)graph G can be found in time $\mathcal{O}(mn \log n)$.*

Important Separators. Roughly speaking, an important separator for sets $X, Y \subseteq V(D)$ is an (X, Y)-separator that cannot be "pushed" towards Y without increasing its size. Formally, this notion is defined as follows.

Definition 2.4.1 (Important Separators)**.** *Let D be a directed graph and $X, Y \subseteq V(D)$. Let $S \subseteq V(D) \setminus (X \cup Y)$ be an (X, Y)-separator and let R be the set of vertices reachable from X in $D - S$. We say that S is an* important (X, Y)-separator *if it is inclusion-wise minimal and there is no (X, Y)-separator $S' \subseteq V(D) \setminus (X \cup Y)$, such that $|S'| \leq |S|$ and $R \subset R'$, where R' is the set of vertices reachable from X in $D - S'$.*

Proposition 2.4.3 ([49])**.** *Let D be a directed graph, $X, Y \subseteq V(D)$ and $k \in \mathbb{N} \cup \{0\}$. Then, D has at most 4^k important (X, Y)-separators of size at most k. In fact, the set of all important (X, Y)-separators in D can be constructed in time $\mathcal{O}(4^k \cdot k^2 \cdot (n + m))$.*

2.5 Treewidth

Roughly speaking, the *treewidth* of an undirected graph is a structural parameter indicating how much a graph resembles a tree. To define treewidth formally, we first need to define the concept of a tree decomposition.

Definition 2.5.1. *A* tree decomposition *of a graph G is a pair (T, β), where T is a rooted tree and $\beta : V(T) \to 2^{V(G)}$, that satisfies the following properties:*

1. **Edge Covering Property:** *For any edge $\{u, v\} \in E(G)$ there exists a node $t \in V(T)$ such that $x, y \in \beta(t)$, and*
2. **Connectivity Property:** *For any vertex $u \in V(G)$, the subgraph of T induced by the set $X_u = \{t \in V(T) : u \in \beta(t)\}$ is a non-empty tree.*

The width *of (T, β) is $\max_{t \in V(T)}\{|\beta(t)|\} - 1$. The* treewidth *of G, denoted by $\mathbf{tw}(G)$, is the minimum width over all tree decompositions of G.*

Given $t, \widehat{t} \in V(G)$, the notation $\widehat{t} \preceq t$ indicates that $\widehat{t}$ is a descendant of t in T. Note that t is a descendant of itself. For any $t \in V(T)$, let t' denote the unique parent of t in T. We also need the standard notations $\sigma(t) = \beta(t) \cap \beta(t')$ and $\gamma(t) = \bigcup_{\widehat{t} \preceq t} \beta(\widehat{t})$.

Proposition 2.5.1 (Folklore). *Let (T, β) be a tree decomposition of a graph G. Given a node $t \in V(T)$, let $t_1, \ldots, t_s$ denote the children of t in T, and for all $i \in [s]$, define $V_{t_i} = \gamma(t_i) \setminus \beta(t)$. Let $V_{t'} = V(G) \setminus (\beta(t) \cup \bigcup_{i=1}^{s} V_{t_i})$. Then, the vertex-set of each connected component of $G \setminus \beta(t)$ is a subset of one of $V_{t_1}, \ldots, V_{t_s}, V_{t'}$.*

The following proposition gives an algorithm to compute a tree decomposition of a particular width of a graph.

Proposition 2.5.2 (Computing a Tree decomposition, [23]). *Given a graph G and $t \in \mathbb{N}$, there is an $\mathcal{O}(t^{\mathcal{O}(t^3)} \cdot n)$-time algorithm that computes a tree decomposition (T, β) of G of treewidth at most t (if such a decomposition exists). Moreover, $|V(T)| = \mathcal{O}(|V(G)|)$.*

We now define a special kind of tree decomposition, called a *nice tree decomposition*.

Definition 2.5.2 (Nice Tree Decomposition). *A tree decomposition (T, β) of a graph G is* nice *if T is a rooted, binary tree with root r, such that $\beta(r) = \emptyset$ and every node $t \in V(T)$ is of the one of the following types.*

- **Leaf***: t is a leaf in T and $\beta(t) = \emptyset$.*
- **Forget***: t has exactly one child, say t', and $\beta(t) = \beta(t') \setminus \{v\}$, where $v \in \beta(t')$.*
- **Introduce***: t has exactly one child, say t', $\beta(t) = \beta(t') \cup \{v\}$, where $v \notin \beta(t')$.*
- **Join***: t has exactly two children, say t_1 and t_2, and $\beta(t) = \beta(t_1) = \beta(t_2)$.*

Whenever we work with a tree decomposition of a graph, we will work with the nice tree decomposition, mainly because the tree T in the nice tree decomposition (T, β) is a rooted, binary tree. Given a tree decomposition (T, β) of a graph G, Bodlaender [23] showed how to construct a *nice* tree decomposition of G of the same width as (T, β).

Proposition 2.5.3 ([23]). *Given a tree decomposition (T, β) of the graph G of width t, a nice tree decomposition (T', β') of G on at most $\mathcal{O}(t \cdot |V(G)|)$ nodes and also of width at most t, can be computed in time $\mathcal{O}(t^2 \cdot \max\{|V(T)|, |V(G)|\})$.*

For a directed graph D, by treewidth of D ($\mathbf{tw}(D)$) and by a (nice) tree decomposition of D, we will refer to the treewidth of the underlying undirected graph of D and a (nice) tree decomposition of the underlying undirected graph of D, respectively. For any $X \subseteq V(D)$, we say that X is a *η-treewidth modulator* if $\mathbf{tw}(D - X) \leq \eta$.

2.6 Branching Algorithms

For all our purposes, a branching algorithm is an ordered list of reduction rules and branching rules. A reduction rule takes an instance of a problem and outputs another equivalent instance of the same problem. Two instances are equivalent only when one is a YES instance if and only if the other is a YES instance. Formally, a reduction rule is a polynomial time procedure replacing an instance $(\mathcal{I}, k)$ of a parameterized language L by a new one $(\mathcal{I}', k')$. It is *safe* if $(\mathcal{I}, k) \in L$ if and only if $(\mathcal{I}', k') \in L$. A branching rule takes an instance $(\mathcal{I}, k)$ of a parameterized language L and produces several instances, $(\mathcal{I}_1, k_1), \ldots, (\mathcal{I}_l, k_\ell)$ of the same problem. A branching rule is *exhaustive* if $(\mathcal{I}, k)$ is a YES instance if and only if at least one of $(\mathcal{I}_1, k_1), \ldots, (\mathcal{I}_l, k_\ell)$ is a YES instance.

The branching algorithm applies the reduction rules and branching rules in the following preference. A branching rule can be applied only when none of the reduction rules are applicable. Reduction rule i can be applied only when none of the reduction rules from 1 to $i-1$ are applicable. Similarly, branching rule i is applicable only when none of the branching rules from 1 to $i-1$ are applicable.

Analysing the running time of branching algorithms - Bounded Search Trees. The running time of a branching algorithm can be analyzed as follows (see, e.g., [65, 82]). Suppose that the algorithm executes a branching rule which has ℓ branching options (each leading to a recursive call with the corresponding parameter value), such that, in the i^{th} branch option, the current value of the parameter decreases by b_i. Then, $(b_1, b_2, \ldots, b_\ell)$ is called the *branching vector* of this rule. We say that α is the *root* of $(b_1, b_2, \ldots, b_\ell)$ if it is the (unique) positive real root of $x^{b^*} = x^{b^*-b_1} + x^{b^*-b_2} + \cdots + x^{b^*-b_\ell}$, where $b^* = \max\{b_1, b_2, \ldots, b_\ell\}$. If $r > 0$ is the initial value of the parameter, and the algorithm (a) returns a result when (or before) the parameter is negative, and (b) only executes branching rules whose roots are bounded by a constant $c > 0$, then its running time is bounded by $\mathcal{O}^*(c^r)$.

Chapter 3

Kernel for DFAS on Bounded Independence Number Digraphs via Reachability Preserving Fault-Tolerant Subgraphs

In this chapter, we design a polynomial kernel for DFAS problem on digraphs of bounded independence number. This result is achieved by establishing a relationship between this problem and the problem of parity reachability fault-tolerance. We begin by discussing the area of fault tolerance below.

3.1 Reachability Preserving Fault-Tolerant Subgraphs

In most real-life applications, even the most reliable networks are highly prone to unexpected failures of a small number of links that connect their nodes. In the past decade, the design of *fault tolerant data structures* for networks has become a central topic of research [15, 17, 40, 177, 174, 22, 43, 44, 45, 21, 77, 176, 175]. Generally, the scenario under study concerns the design of a structure that, after the failure of any set F of at most $k \geq 1$ arcs (representing links) in a given digraph D (representing a network), should provide a fast answer to certain types of queries that address the properties of $D - F$. The most common queries of this form address the *reachability* between two vertices, or, more generally, the length of a shortest path existent, if any, between them. Indeed, reachability (or, more generally, distance preservation)

is the most basic requirement to maintain to ensure that the network functions properly. In this context, particular attention has been given to the case where the data structure should consist of a subgraph or a minor of D with as fewest arcs/vertices as possible [15, 177, 17, 21, 16, 176, 40]. Then, queries can be answered by standard means as the usage of BFS or Dijkstra's algorithm. In particular, these simple data structures are of interest as they also double as *sparsifiers*. The study of various graph sparsifiers—such as *flow-sparsifiers* [139] which are closely related to the aforementioned data structures—is a fundamental, active area of research in computer science and structural graph theory [61, 8, 87, 139, 42].

More concretely, in the Fault-Tolerance (S,T)-Reachability problem (or FTR(S,T) for short), we are given a digraph D, two (not necessarily disjoint) terminals sets $S,T \subseteq V(D)$, and a positive integer k. The objective is to construct a subgraph H of D with minimum number of arcs/vertices such that, after the failure of any set of at most k arcs in D, the following property is preserved for any two vertices $s \in S$ and $t \in T$: if there still exists a directed path from s to t in D, then there also still exists a directed path from s to t in H. Clearly, a trivial lower bound on the number of arcs in H is $m = \mathcal{O}(n^2)$, where $n = |V(D)|$ and $m = |E(D)|$. For the case where $|S| = 1$ and $T = V(D)$, Baswana et al. [17] presented a construction of a subgraph H with $\mathcal{O}(2^k n)$ arcs in time $\mathcal{O}(2^k nm)$. Additionally, they gave a tight matching lower bound: for any $n, k \in \mathbb{N}$ where $n \geq 2^k$, there exists a digraph on n vertices where H must have $\Omega(2^k n)$ arcs.

Naturally, the question of the improvement of the dependency on k arises for special classes of digraphs. However, an arguably more radical research direction to pursue concerns the dependency on n.

> Which are the largest classes of digraphs for which FTR(S,T) admits subgraphs whose size dependency on n can be made sublinear, logarithmic *or even constant*?

At first glance, when we consider the simplest sparsest digraph existent, this pursuit seems futile. Indeed, already in the case where $S = \{s\}$, $T = \{t\}$, $k = 1$ and D is a directed path from s to t, the only solution is to choose $H = D$. At second glance, when we consider the simplest densest digraph existent, again we reach a dead-end: for S, T and k as before, define D as the tournament obtained by adding, to a directed path $s = v_1 \to v_2 \to \ldots \to v_n = t$, all arcs going from v_i to v_j for every $j + 1 < i$; then, to construct H, we must select the entire path.

We show that "almost acyclicity" suffices to eliminate the dependency on n *entirely* for a broad class of dense digraphs called *bounded independence number digraphs*. Furthermore, one can achieve a polynomial dependence in terms of k for this digraph class.

To step beyond the strict confinement of tournaments where *all* relations (arcs) between the input entities (vertices) must be both *present* and *known*, Fradkin and Seymour [106] initiated the study of bounded independence digraphs. Formally, for any integer $\alpha \geq 1$, the class of α-bounded independence digraphs, denoted by $\mathcal{D}_\alpha$, is defined as follows. Let $max\text{-}is(D)$ denote the size of a maximum independent set in D.

$$\mathcal{D}_\alpha = \{D \mid D \text{ is a digraph and } max\text{-}is(D) \leq \alpha\}.$$

For this class of digraphs, Fradkin and Seymour [106] studied the k-Disjoint Paths problem, and showed that it admits a polynomial time algorithm for any fixed value of k. Observe that $\mathcal{D}_\alpha$ is *hereditary*, and for $\alpha = 1$, it coincides with the class of tournaments. Furthermore, even for $\alpha = 2$, it contains digraphs with a linear fraction of vertex pairs that have no arc between them—thus, it can accommodate the lack of a large number of links/relations.

Our main technical contribution is the following combinatorial lemma.

Lemma 3.1.1. *Given a digraph $D \in \mathcal{D}_\alpha$, positive integers k and ℓ, and $S \subseteq V(D)$ such that every strongly connected component of $D - S$ has at most ℓ vertices, the* Fault-Tolerance (S, S)-Reachability (FTR(S, S)) *problem admits a solution H on $|S|^2(k\ell)^{\mathcal{O}(4^{\alpha}\ell^2)}$ vertices. Furthermore, such a solution H can be found in polynomial time.*

In particular, when $D - S$ is acyclic, $\ell = 1$. Thus, if $|S|$ and ℓ are independent of n (such as the case where $|S| = |T| = \ell = 1$ discussed earlier), the dependency on n is eliminated. (We remark that a solution for Fault-Tolerance (S, T)-Reachability where $S \neq T$ is subsumed by a solution for Fault-Tolerance $(S \cup T, S \cup T)$-Reachability.) Note that we extend the class of digraphs dealt with beyond acyclicity at two fronts: enabling S to be a modulator, thus $D-S$ rather than D should be "almost acyclic"; enabling the strongly connected components to be of size that is ("small" but) larger than 1.

Based on our combinatorial lemma (Lemma 3.1.1), we establish the following theorem.

Theorem 3.1.1. DFAS *on $\mathcal{D}_\alpha$ admits a kernel of size $k^{\mathcal{O}(4^{\alpha})}$.*

3.2 Introducing Cut Preserving Sets

The most central notion in this chapter is of a *cut preserving set*. Informally, for a digraph D, a pair of vertices s, t and an integer k, a set $\mathcal{Z} \subseteq V(D)$ is called a k-cut preserving set[1] for (s, t) in D if it preserves all (s, t)-arc cuts of size at most k. That is, A is an (s, t)-arc cut with at most k arcs in D if and only if A is a such a cut in $D[\mathcal{Z}]$. Observe that the graph induced on such a k-cut preserving set $\mathcal{Z}$ is a candidate solution for FTR($\{s\}, \{t\}$) problem. Clearly $V(D)$ is a k-cut preserving set for any pair of vertices s, t. The intent is to have such a set of "small" size. Towards this, let us discuss some properties that suffice for $\mathcal{Z}$ to be a k-cut preserving set for (s, t) in D.

Since $\mathcal{Z} \subseteq V(D)$, any (s, t)-arc cut of D is an (s, t)-arc cut of $D[\mathcal{Z}]$. For the other direction, we need the property that, for any $A \subseteq E(D)$ of size at most k, the existence of an (s, t)-path in $D-A$ implies the existence of an (s, t)-path in $D[\mathcal{Z}]-A$. Let us now see which properties suffice to imply the above property. We begin with a special case. Suppose there is a "large" flow from s to t in D. In particular, suppose there are at least $k + 1$ internally vertex-disjoint (s, t)-paths in D. Then, in $\mathcal{Z}$ it is enough to keep the vertices of some $k + 1$ vertex-disjoint (s, t)-paths, as no arc set of size at most k can hit all these paths. The more involved case occurs when the flow from s to t in D is at most k. Consider any (s, t)-path P in D. Ideally (if we did not have a size constraint on $\mathcal{Z}$) we would have preserved all the vertices of P in $\mathcal{Z}$. Clearly, this can be expensive in terms of the size of $\mathcal{Z}$. Nevertheless, we can merge the ideas above (the "large-flow idea" and the "keep-full-path idea") to get the desired result. To see this, let P be a (s, t)-path in D. Let $\mathcal{Z}$ be a set of vertices such that, either all the vertices of P are in $\mathcal{Z}$ or if the vertices of a (u, v)-subpath of P are not in $\mathcal{Z}$, then there are $k + 1$ internally vertex-disjoint (u, v)-paths in $D[\mathcal{Z}]$. That is, if the vertices of a subpath are missing in $\mathcal{Z}$, then $\mathcal{Z}$ contains a witness of a large flow for the endpoints of this subpath. Observe that such a set $\mathcal{Z}$ suffices to be a k-cut preserving set for (s, t) in D. This is because if P is an (s, t)-path in $D - A(A \subseteq E(D)$ and $|A| \leq k)$, then either all the vertices of P are in $\mathcal{Z}$ or for any missing (u, v)-subpath of P, since there are $k+1$ vertex-disjoint (u, v)-paths in $D[\mathcal{Z}]$, at least one still remains in $D[\mathcal{Z}]-A$. Thus, in $D[\mathcal{Z}]-A$, one can find an (s, t)-path: for the missing subpaths of P in $\mathcal{Z}$, there exists some (other) path between the same endpoints in $D[\mathcal{Z}] - A$ which together yield an (s, t)-walk (and hence an (s, t)-path) in $D[\mathcal{Z}] - A$. These properties are formalized in Definition 3.4.1.

[1]This is not the way it is defined later. However, for the sake of exposition, we start with this definition and refine it to have properties that also guarantee this property implicitly.

3.2.1 Computing k-Cut Preserving Sets: The Ideas

Next we give an intuition for how one can compute such k-cut preserving sets for a digraph $D \in \mathcal{D}_\alpha$, each of whose strongly connected component has size at most ℓ. For exposition purposes, consider (for now), only the case where D is acyclic (i.e., $\ell = 1$). With a certain technical argument, the general case reduces to this one. Moreover, we use the definition of a k-cut preserving set from the beginning of this section for this illustration as it allows us to convey our ideas in a clearer manner.

The proof will use induction on α. As the base case, consider the case when $\alpha = 1$, that is, D is a transitive tournament. As D is transitive, there exists a topological ordering of the vertices of D. Consider the set S of vertices between s and t in this ordering. Note that any path from s to t only uses vertices in S. So, either S is smaller than $k+1$, and then $S \cup \{s,t\}$ is a k-cut preserving set for (s,t), or it can be seen that there is no arc-cut for (s,t) of size at most k. In the latter case, the union of $\{s,t\}$ and any subset of $k+1$ vertices of S is a k-cut preserving set for (s,t); indeed, in the subgraph induced by the union there is still no arc-cut for (s,t) of size at most k.

Now, let us hint at how the inductive step of the proof works. First, we note that, if $P_1, \ldots, P_{k+1}$ are $k+1$ internally vertex-disjoint (s,t)-paths, then $\mathcal{Z} = \cup_{i \in [k+1]} P_i$ is a k-cut preserving set for the pair (s,t), because there is no arc-cut of (s,t) in both D and $D[\mathcal{Z}]$ of size at most k. Moreover, since D is acyclic and $D \in \mathcal{D}_\alpha$, if these paths exist, then Observation 3.3.1 implies that we can assume that all these paths are shorter than $2\alpha + 1$ and thus $|\mathcal{Z}| \leq k(2\alpha+1)$.

The last argument means that we can assume the existence of a (s,t)-vertex cut of size at most k. For simplicity, suppose that $\{c_1, c_2\}$ is a minimal (s,t)-vertex -cut. Since $\{c_1, c_2\}$ is a vertex cut, any path from s to t in D can be decomposed as a path from s to c_i, a path from c_i to c_j and then a path from c_j to t, where i and j are two indices (possibly equal) in $\{1,2\}$. Here, we mean that none of the three paths contains c_i (or c_j) as an internal vertex. For $i \in \{1,2\}$, let S_i be the union of the set of vertices of the paths from s to c_i that intersect $\{c_1, c_2\}$ only on the last vertex, and T_i be the union of the set of vertices of the paths from c_i to t that intersect $\{c_1, c_2\}$ only on the first vertex. Finally, for distinct $i, j \in \{1,2\}$, let $C_{i,j}$ be the union of the set of vertices of the paths from c_i to c_j. Because of the last remark on how any path from s to t can be decomposed, taking the union of six k-cut preserving sets-namely, for each $i, j \in \{1,2\}$, $i \neq j$, for (s, c_i) in $D[S_i]$, (c_i, t) in $D[T_i]$ and (c_i, c_j) in $D[C_{i,j}]$- gives a k-cut preserving set for (s,t) in D.

Now, the question is how to use the induction hypothesis to find a k-cut preserving set for each of these pairs. Consider first the digraph induced by the vertices in S_1. Because $\{c_1, c_2\}$ is a minimal (s,t)-vertex cut, the only vertices of S_1 that can possibly have "outgoing arcs towards" t in S_1 are s and c_1. Moreover, since $\{c_1, c_2\}$ is a minimal (s,t)-vertex cut, there exists a path from c_1 to t in D and thus t is reachable from any vertex of S_1. However, since D is acyclic, this means that there is no arc from t to any of the vertices of S_1, else we would get a closed walk and thus a cycle. This implies that $D[S_1 \setminus \{s, c_1\}] \in \mathcal{D}_{\alpha-1}$ as any independent set of $S_1 \setminus \{s, c_1\}$ can be extended with t. We cannot apply the induction hypothesis to find a k-cut preserving set for (s, c_1) in S_1 because the independence number of $D[S_1]$ could be equal to α, however the above shows the spirit of the arguments that will be used to find subgraphs with smaller independence number where we can apply the induction hypothesis. A similar argument would also give that the independence number of $D[T_1 \setminus \{c_1, t\}]$ is at most $\alpha - 1$ as any independent set can be extended using s.

The previous argument does not apply to $C_{1,2}$, because the vertices of $C_{1,2}$ can be adjacent to s or t (some vertices of $C_{1,2}$ can be adjacent to s and some can be adjacent to t). This is the case that requires a stronger and more technical definition for a k-cut preserving set. In particular, we need to understand what happens to the vertices of D that are on a path from s to t but do not belong to a k-cut preserving set for this pair.

In the next section we give some notations and simple results and observations that will be used to give the technical details of this chapter.

3.3 Prelude to the Technical Details

For a digraph D and $X \subseteq V(D)$, we say that a (u,v)-path P in D is *X-free* if none of the internal vertices of P are from X. The *X-based partition of P* in D is the partition $P = P_1 \circ \ldots \circ P_q$ such the union of the end-points of P_i, $i \in [q]$, is exactly the set $(X \cap V(P)) \cup \{u, v\}$. A *semi-$X$-based partition* of P, $P = P_1 \circ \ldots \circ P_q$, is such that the end-points of the paths P_i, $i \in [q]$, are a subset of $(X \cap V(P)) \cup \{u, v\}$.

Vertex and Arc Cuts: For a digraph D and $u, v \in V(D)$, a *(u,v)-arc cut* is a set of arcs of D, say X, such that $D - X$ has no (u,v)-path. A *(u,v)-vertex cut* is a set of vertices of D, say Y, such that $D - Y$ has no (u,v)-path and $u, v \notin Y$ if $(u,v) \notin E(D)$.

We now give some simple results concerning the class $\mathcal{D}_\alpha$ that will be used throughout this chapter and the next chapter.

Observation 3.3.1. *Let $D \in \mathcal{D}_\alpha$. The length of the shortest cycle in D is at most $2\alpha + 1$. Also, the length of any induced path in D is at most $2\alpha + 1$.*

Lemma 3.3.1. *If $D \in \mathcal{D}_\alpha$, then $|E(D)| \geq \frac{n}{2}(\frac{n}{\alpha} - 1)$.*

Proof. The proof follows from Turan's theorem [76], which states that any graph on n vertices that does not contain a clique of size $\alpha + 1$ has at most $(1 - \frac{1}{\alpha})\frac{n^2}{2}$ edges. Thus the number of edges in D is at least $\frac{n^2}{2} - (1 - \frac{1}{\alpha})\frac{n^2}{2} = \frac{n}{2}(\frac{n}{\alpha} - 1)$. □

3.4 Finding Small k-Cut Preserving Sets

We give the precise definition of a k-cut preserving set.

Definition 3.4.1 (k-Cut Preserving Set)**.** *For digraph D, an ordered pair (u, v) of vertices of D and a positive integer k, $\{u, v\} \subseteq \mathcal{Z} \subseteq V(D)$ is a* k-cut preserving set for (u, v) in D *if the following holds. For any (u, v)-path P in D, there exists a semi-$\mathcal{Z}$-based partition $P_1 \circ \ldots \circ P_d$ of P with the following two properties. For each $i \in [d]$, P_i is an (s_i, t_i)-path in D with $s_i, t_i \in \mathcal{Z}$. Moreover, either $V(P_i) \subseteq \mathcal{Z}$ or there exists a list $\mathcal{L}_i$ of $k+1$ internally vertex-disjoint (s_i, t_i)-paths in $D[\mathcal{Z}]$. A list $\mathcal{L}_i$ with the above property is called a* replacement kit for P_i in $\mathcal{Z}$. *Such a semi-$\mathcal{Z}$-based partition of P is called a* $\mathcal{Z}$-replacement witness for P.

Before moving to the computational aspects of a k-cut preserving set, we give the following lemma that can be considered as the main utility of k-cut preserving sets, and relate to the intuition we gave in the previous section.

Lemma 3.4.1. *Let D be a digraph, $u, v \in V(D)$ and $\mathcal{Z}$ be a k-cut preserving set for (u, v) in D. For any set $A \subseteq E(D)$ of at most k arcs, if there exists a (u, v)-path in $D - A$, then there also exists one in $D[\mathcal{Z}] - A$.*

Proof. Consider some $A \subseteq E(D)$ such that $|A| \leq k$. Suppose there exists a (u, v)-path P in $D - A$. Since $\mathcal{Z}$ is a k-cut preserving set for the pair (u, v), there exists a semi-$\mathcal{Z}$-based partition $P = P_1 \circ \ldots \circ P_d$ such that for each $j \in [d]$, P_j is an (s_j, t_j)-path, $s_j, t_j \in \mathcal{Z}$ and, either $V(P_j) \subseteq \mathcal{Z}$, in which case P_j is a path in $D[\mathcal{Z}] - A$, or there exist $k + 1$ internally vertex- disjoint (s_j, t_j)-paths in $D[\mathcal{Z}]$. In the later case,

at least one of the $k+1$ paths is in $D[\mathcal{Z}] - A$ (because $|A| \leq k$). This implies the existence of a walk from u to v (and hence also a (u,v)-path) in $D[\mathcal{Z}] - A$. This concludes the proof. □

The main goal of this section is to prove the following lemma.

Lemma 3.4.2 (k-Cut Preserving Lemma)**.** *Let D be an acyclic digraph, and $u, v \in V(D)$ be such that $N^-(u) = N^+(v) = \emptyset$. Additionally, let $D - \{u,v\} \in \mathcal{D}_\alpha$. Then there exists a k-cut preserving set for (u,v) in D of size at most $f(\alpha)$, where $f(1) = k^3 + 5k^2 + 3k$ and for $\alpha > 1$, $f(\alpha) = k^2 g(\alpha) + 2kh(\alpha)$, $g(\alpha) = (2k + (k + kf(\alpha - 1))^2) f(\alpha - 1)$ and $h(\alpha) = (k^2 + k)g(\alpha) + kf(\alpha - 1)$. Moreover, such a set can be found in time $n^{\mathcal{O}(1)}$, where $n = |V(D)|$.*

Note that $V(D)$ is always a k-cut preserving set for any pair of vertices (u,v) in D, for any k. We now define a notation, for the sake of convenience, that will be used throughout this section. For any digraph D, $u, v \in V(D)$ and $X \subseteq V(D)$, let $ver_D(u,v;X)$ denote the union of the sets of vertices of all X-free (u,v)-paths in D. Observe that $ver_D(u,v;X) \cap X \subseteq \{u,v\}$. We begin by making an observation that forms the base line for computing small sized k-cut preserving sets using an appropriate induction.

Observation 3.4.1. *Let D be a digraph, $u, v \in V(D)$, $\mathcal{Z} \subseteq V(D)$ and k be a positive integer. Let P be a (u,v)-path in D, and $P = P_1 \circ \ldots \circ P_d$ be a semi-$\mathcal{Z}$-based partition of P. If for each $i \in [d]$, there is a $\mathcal{Z}_i$-replacement witness for P_i in D_i, for some $\mathcal{Z}_i \subseteq \mathcal{Z}$ and D_i subgraph of D, then there is a $\mathcal{Z}$-replacement witness for P.*

Proof. For each $i \in [d]$, let $P_I = P_{i,1} \circ \ldots \circ P_{i,c_i}$ be a $\mathcal{Z}_i$-replacement witness for $\mathcal{Z}_i$ in D_i. Then, consider the semi-$\mathcal{Z}$-based partition $P = P_{1,1} \circ \ldots \circ P_{1,c_1} \circ P_{2,1} \circ \ldots \circ P_{2,c_2} \circ \ldots \circ P_{d,1} \circ \ldots \circ P_{d,c_d}$. Then, for each $i \in [d]$ and $j \in [c_i]$, either $V(P_{i,j}) \subseteq \mathcal{Z}_i \subseteq \mathcal{Z}$, or there exists a list $\mathcal{Z}_{i,j}$ containing $k+1$ internally vertex-disjoint $(V(D_i) \setminus \mathcal{Z}_i)$-free $(x_{i,j}, y_{i,j})$-paths in D_i such that $P_{i,j}$ is a $(x_{i,j}, y_{i,j})$-path. Since $\mathcal{Z}_i \subseteq \mathcal{Z}$ and D_i is a subgraph of D, the paths in $\mathcal{L}_{i,j}$ are $(V(D) \setminus \mathcal{Z})$-free and exist in D. □

Next, we give two lemmas (Lemmas 3.4.3 and 3.4.4) that basically use Observation 3.4.1 in a more concrete setting required to prove the k-Cut Preserving Lemma by induction on the size of the maximum independent set in the digraph.

Lemma 3.4.3. *Let D be a digraph, $u, v \in V(D)$ and k be a positive integer. Let C be some (u,v)-vertex cut in D. For each $c \in C$, let $\mathcal{Z}(u,c)$ (resp. $\mathcal{Z}(c,v)$) be a k-cut*

preserving set for (u,c) (resp. (c,v)) in $D[ver_D(u,c;C)]$ (resp. $D[ver_D(c,v;C)]$). For each $(c,c') \in C^2$, $c \neq c'$, let $\mathcal{Z}(c,c')$ be a k-cut preserving set for (c,c') in $D[ver_D(c,c';C)]$. Then, $\mathcal{Z} := \bigcup_{c\in C}(\mathcal{Z}(u,c) \cup \mathcal{Z}(c,v)) \cup \bigcup_{(c,c')\in C^2, c\neq c'} \mathcal{Z}(c,c')$ is a k-cut preserving set for (u,v) in D.

Proof. First observe, from the definition of a k-cut preserving set and the construction of $\mathcal{Z}$, that $C \subseteq \mathcal{Z}$. Consider any (u,v)-path P in D. Let $P = P_1 \circ \ldots \circ P_q$ be the C-based partition of P. Since $C \subseteq \mathcal{Z}$, $P_1 \circ \ldots \circ P_q$ is a semi-$\mathcal{Z}$-based partition of P. Then P_1 is a C-free (u,c_1)-path in D for some $c_1 \in C$, P_q is a C-free (c_2,v)-path in D for some $c_2 \in C$, and for each $i \in [2,q-1]$, P_i is a C-free $(c_{j_i}, c_{j_i'})$-path in D, for some $c_{j_i}, c_{j_i'} \in C$, $j_i \neq j_i'$. Thus, P_1 is a (u,c_1)-path in $D[ver_D(u,c_1;C)]$, P_q is a (c_2,v)-path in $D[ver_D(c_2,v;C)]$, and for each $i \in [2,q-1]$, P_i is a $(c_{j_i}, c_{j_i'})$-path in $D[ver_D(c_{j_i}, c_{j_i'};C)]$. Since $\mathcal{Z}(u,c_1), \mathcal{Z}(c_2,v), \cup_{i\in[2,q-1]}\mathcal{Z}(c_{j_i}, c_{j_i'}) \subseteq \mathcal{Z}$, we are done by Observation 3.4.1. □

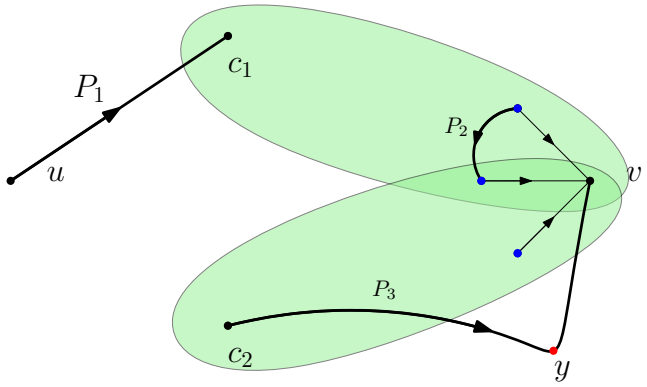


Figure 3.1: (c_1, c_2) is a (u,v) vertex-cut, the green parts correspond to the $\mathcal{Z}(c_i,v)$ and the blue vertices are the vertices of X. P_1 is a path of Type $(u,\square)$, P_2 is a path of Type $(\boxtimes,\boxtimes)$ and P_3 is a path of Type $(\square,\boxminus,v)$ with $y \in Y$.

Lemma 3.4.4. *Let D be a digraph, $u,v \in V(D)$, and k be a positive integer. Let C be some (u,v)-vertex cut in D. For each $c \in C$, let $\mathcal{Z}(u,c)$ (resp. $\mathcal{Z}(c,v)$) be a k-cut preserving set for (u,c) (resp. (c,v)) in $D[ver_D(u,c;C)]$ (resp. $D[ver_D(c,v;C)]$). Let $X = N_D^-(v) \cap \bigcup_{c\in C} \mathcal{Z}(c,v)$. For each $(\mathtt{a},\mathtt{b}) \in (C\cup X)^2$, $\mathtt{a} \neq \mathtt{b}$, let $\mathcal{Z}(\mathtt{a},\mathtt{b})$ be a k-cut preserving set for $(\mathtt{a},\mathtt{b})$ in $D[ver_D(\mathtt{a},\mathtt{b};C \cup N_D^-(v))]$. Then, $\mathcal{Z} := \bigcup_{c\in C}(\mathcal{Z}(u,c) \cup \mathcal{Z}(c,v)) \cup \bigcup_{(\mathtt{a},\mathtt{b})\in(C\cup X)^2,\mathtt{a}\neq\mathtt{b}} \mathcal{Z}(\mathtt{a},\mathtt{b})$ is a k-cut preserving set for (u,v) in D.*

Proof. First observe that $\{u,v\} \cup C \cup X \subseteq \mathcal{Z}$. Let $Y = N_D^-(v) \setminus X$. We begin by defining some special types of paths (see Figure 3.1).

1. A path P *is of Type* $(u, \square)$ *(resp.* $(\square, v)$*)* if it is a C-free (u, c)-path (respectively (c, v)-path) in D for some $c \in C$.

2. A path P *is of Type* $(\boxtimes, \boxtimes)$ if it is a $(C \cup N_D^-(v))$-free $(\mathtt{a}, \mathtt{b})$-path in D for some $(\mathtt{a}, \mathtt{b}) \in (C \cup X)^2$.

3. A path P *is of Type* $(\square, \boxminus, v)$ if it is a (c, v)-path in D for some $c \in C$ and there exists $y \in V(P) \cap Y$ such that the (c, y)-subpath of P is C-free.[2]

We now begin with the proof of the lemma. Let P be some (u, v)-path. We need to show that there is a $\mathcal{Z}$-replacement witness for P. Let $P = P'_1 \circ \ldots \circ P'_q$ be the $(C \cup X)$-based partition of P. If P is not Y-free, that is, $V(P) \cap Y \neq \emptyset$, let $s' \in [q]$ be the least integer such that $V(P'_{s'}) \cap Y \neq \emptyset$. If P is Y-free, let $s' = q$. Let $s \leq s'$ be the largest integer such that P_s is an $(\mathtt{a}, \mathtt{b})$-path, where $\mathtt{a} \in C$ and $\mathtt{b} \in C \cup X \cup \{v\}$. We first show that such an s always exists. From the definition of s', either there exists some $y \in Y$ in $V(P'_{s'})$ or $v \in V(P'_{s'})$. In the later case, since C is a (u, v)-vertex cut, there exists $c \in C$ such that c appears on P. Since $P = P'_1 \circ \ldots \circ P'_q$ is a $C \cup X$-based partition of P, there exists $s \leq s'$ such that P_s is a $(\mathtt{a}, \mathtt{b})$-path where $\mathtt{a} \in C$. In the former case again, since $y \in Y \subseteq N_D^-(v)$ and C is a (u, v)-vertex cut using previous arguments the existence of the desired s is guaranteed.

Consider the partition $P = P_1 \circ \ldots \circ P_s$, such that $P_i = P'_i$, if $i < s$ and $P_s = P'_s \circ P'_{s+1} \circ \ldots \circ P'_q$. Observe that, since $C \cup X \subseteq \mathcal{Z}$, $P = P_1 \circ \ldots \circ P_s$ is a semi-$\mathcal{Z}$-based partition of P.

Claim 3.4.1. *P_1 is a Type $(u, \square)$ path, for each $i \in [2, s-1]$, P_i is a Type $(\boxtimes, \boxtimes)$ path and, P_s is either a Type $(\square, v)$ or Type $(\square, \boxminus, v)$ path.*

Proof. Recall that $P = P'_1 \circ \ldots \circ P'_q$ is the $(C \cup X)$-based partition of P. Thus, we have the following.

1. For each $i \in [q]$, P'_i is $(C \cup X)$-free path.

2. For each $i \in [2, q-1]$, P'_i is a $(\mathtt{a}, \mathtt{b})$-path, where $(\mathtt{a}, \mathtt{b}) \in (C \cup X)^2$.

3. Since C is a (u, v)-vertex cut in D and $X \subseteq N_D^-(v)$, P'_1 is a (u, c)-path for some $c \in C$.

[2]Specifically, if there exists $y \in V(P) \cap Y$ with this property, then the first vertex of P that belongs to Y also has that property.

4. From the choice of s, for each $i \in [s-1]$, $V(P'_i) \cap Y = \emptyset$. Since for $i \in [s-1]$, $P_i = P'_i$ and $X \cup Y = N_D^-(v)$, P_i is $(C \cup N_D^-(v))$-free.

Thus, from Points 2 and 4, for each $i \in [s-1]$, P_i is of Type $(\boxtimes, \boxtimes)$. Also, from Points 3 and 4, P_1 is of Type $(u, \square)$. We now show that P_s is of Type $(\square, v)$ or $(\square, \boxminus, v)$. From the choice of s and the construction of P_s, P_s is a (c, v)-path for some $c \in C$. If P is Y-free, then P_s is of Type $(\square, v)$, otherwise, P_s is of Type $(\square, \boxminus, v)$. $\square$

For each $i \in [s]$, define $\mathcal{Z}_i$ and D_i as follows.

$$\mathcal{Z}_i = \begin{cases} \mathcal{Z}(u, c) & \text{if } i = 1,\ P_1 \text{ is a } (u, c)\text{-path}, c \in C \\ \mathcal{Z}(\mathtt{a}, \mathtt{b}) & \text{if } i \in [2, s-1],\ P_i \text{ is a } (\mathtt{a}, \mathtt{b})\text{-path}, (\mathtt{a}, \mathtt{b}) \in (C \cup X)^2 \\ \mathcal{Z}(c, v) & \text{if } i = s,\ P_s \text{ is a } (c, v)\text{-path}, c \in C \end{cases}$$

$$D_i = \begin{cases} D[ver_D(u, c; C)] & \text{if } i = 1, P_1 \text{ is a } (u, c)\text{-path}, c \in C \\ D[ver_D(\mathtt{a}, \mathtt{b}; (C \cup N_D^-(v))] & \text{if } i \in [2, s-1],\ P_i \text{ is a } (\mathtt{a}, \mathtt{b})\text{-path}, (\mathtt{a}, \mathtt{b}) \in (C \cup X)^2 \\ D[ver_D(c, v)] & \text{if } i = s,\ P_s \text{ is a } (c, v)\text{-path}, c \in C \end{cases}$$

Recall the construction of $\mathcal{Z}$ from the lemma statement. Observe that for each $i \in [s]$, $\mathcal{Z}_i \subseteq \mathcal{Z}$. From Observation 3.4.1, to give a $\mathcal{Z}$-replacement witness for P, it is enough to give a $\mathcal{Z}_i$-replacement witness for each P_i, in D_i, $i \in [s]$. Thus, the following claim will finish the proof of the lemma.

Claim 3.4.2. *For each $i \in [s]$, P_i has a $\mathcal{Z}_i$-replacement witness in D_i.*

Proof. We prove the claim using the following cases.

- **Case $i = 1$:** From Claim 3.4.1, P_1 is a C-free (u, c)-path in D for some $c \in C$. Thus, P_1 is a (u, c)-path in D_1. Since Z_1 is a k-cut preserving set for (u, c) in D_i, there exists a $\mathcal{Z}_1$-replacement witness for P_1 in D_1.

- **Case $i \in [2, s-1]$:** From Claim 3.4.1, when $i \in [2, s-1]$, then P_i is a $(C \cup N_D^-(v))$-free $(\mathtt{a}, \mathtt{b})$-path in D for some $(\mathtt{a}, \mathtt{b}) \in (C \cup X)^2$. Thus, P_i is an

$(\mathtt{a}, \mathtt{b})$-path in D_i. Since Z_i is a k-cut preserving set for $(\mathtt{a}, \mathtt{b})$ in D_i, there exists a $\mathcal{Z}_i$-replacement witness for P_i in D_i.

- **Case** $i = s$**:** From Claim 3.4.1, P_s is of either Type $(\square, v)$ or Type $(\square, \boxminus, v)$.

  - P_s **is of Type** $(\square, v)$**:** From the definition of Type $(\square, v)$, P_s is a C-free (c, v)-path in D, for some $c \in C$. Thus, P_s is a (c, v)-path in D_s. Since $\mathcal{Z}_s$ is a k-cut preserving set for (c, v) in D_s, there exists a $\mathcal{Z}_s$-replacement witness for P_s in D_s.

  - P_s **is of Type** $(\square, \boxminus, v)$**:** From the definition of Type $(\square, \boxminus, v)$, P_s is a (c, v)-path in D, for some $c \in C$, and there exists $y \in V(P) \cap Y$ such that the (c, y)-subpath of P is C-free. Let $P_s^\dagger$ be the (c, y)-subpath of P. Recall that $Y = N_D^-(v) \setminus X$. Consider the (c, v)-path in D, denoted by $\widetilde{P_s}$, obtained by appending the arc (y, v) at the end of $P_s^\dagger$. That is, $\widetilde{P_s} = P_s^\dagger \circ (y, v)$. Since $P_s^\dagger$ is a C-free path, so is $\widetilde{P_s}$. Thus $\widetilde{P_s}$ is a (c, v)-path in D_s. Since $\mathcal{Z}_s$ is a k-cut preserving set for (c, v) in D_s, there exists a semi-$\mathbb{Z}_s$-based partition of $\widetilde{P_s}$ which is a $\mathcal{Z}_s$-replacement witness for $\widetilde{P_s}$ in D_s. Let $\widetilde{P_s} = \widetilde{P_{s,1}} \circ \ldots \circ \widetilde{P_{s,r}}$ be one such partition. Since $y \in Y = N_D^-(v) \setminus X$ and $\mathcal{Z}_s \subseteq X$, $y \notin \mathcal{Z}_s$. Thus, y is an internal vertex of $\widetilde{P_{s,r}}$. Let $\widetilde{P_{s,r}}$ be an (x, v)-path. Clearly, $x \in \mathcal{Z}_s$ because $\widetilde{P_s} = \widetilde{P_{s,1}} \circ \ldots \circ \widetilde{P_{s,r}}$ is a semi-$\mathcal{Z}_s$-based partition. Let $P_{s,r}^\dagger$ be the (x, v)-subpath of $P_{s,r}$. We claim that $P_s = \widetilde{P_{s,1}} \circ \ldots \circ \widetilde{P_{s,r-1}} \circ P_{s,r}^\dagger$ is a semi-$\mathcal{Z}_s$-based partition of P_s and is also a $\mathcal{Z}_s$-replacement witness for P_s in D_s. It is clear from the discussion above that $P_s = \widetilde{P_{s,1}} \circ \ldots \circ \widetilde{P_{s,r-1}} \circ P_{s,r}^\dagger$ is a semi $\mathcal{Z}_s$-based partition of P_s. We will now show that it is a $\mathcal{Z}_s$-replacement witness for P_s in D_s.

    Since $\widetilde{P_s} = \widetilde{P_{s,1}} \circ \ldots \circ \widetilde{P_{s,r}}$ is a $\mathcal{Z}_s$-replacement witness for $\widetilde{P_s}$, we have that for each $j \in [r]$, either $V(\widetilde{P_{s,j}}) \subseteq \mathcal{Z}_s$ or there exists a list $\mathcal{L}_j$ containing $k+1$ vertex disjoint paths from the start vertex of $\widetilde{P_{s,j}}$ to its end vertex. Also, since $y \notin \mathcal{Z}_s$ and y is an internal vertex of $\widetilde{P_{s,r}}$, $V(\widetilde{P_{s,r}}) \not\subseteq \mathcal{Z}_s$. Thus, there is a list $\mathcal{L}_r$ containing $k+1$ vertex disjoint (x, v)-paths (recall x and v are the start and end vertices, respectively, of $\widetilde{P_{s,r}}$). Since $P_s = \widetilde{P_{s,1}} \circ \ldots \widetilde{P_{s,r-1}} \circ P_{s,r}^\dagger$, and $P_{s,r}^\dagger$ is an (x, v)-path, from the above discussion for each $j \in [r-1]$, either $V(\widetilde{P_{s,j}}) \subseteq \mathcal{Z}_s$ or there exists a list $\mathcal{L}_j$ containing $k+1$ vertex disjoint paths from the start vertex of $\widetilde{P_{s,j}}$ to its end vertex. Also, there exists a list, $\mathcal{L}_r$, containing $k+1$ vertex disjoint paths from the start vertex of $P_{s,r}^\dagger$ to its end vertex. This completes the proof of the claim.

□

As argued earlier, this completes the proof of the lemma. □

3.4.1 Finding a Small k-Cut Preserving Set for a Pair with a Large Flow

As explained in Section 3.2, the proof of Lemma 3.4.2 will distinguish whether there is a k vertex-cut for (s,t) or not. The case where there is a no k vertex-cut is the easiest one, and will be dealt with the following lemma by simply keeping $k+1$ vertex disjoint paths.

Lemma 3.4.5. *Let $D \in \mathcal{D}_\alpha$ be an acyclic digraph and $u, v \in V(D)$ be such that each (u,v)-vertex cut in D has size at least $k+1$. Then, a k-cut preserving set for (u,v) in D of size at most $(2\alpha-1)(k+1)+2$ exists and is computable in $n^{\mathcal{O}(1)}$ time, where $n = |V(D)|$.*

Proof. Since every (u,v)-vertex cut in D has size at least $k+1$, from Menger's Theorem, there are at least $k+1$ vertex-disjoint (u,v)-paths in D. Let $Q'_1, \ldots, Q'_{k+1}$ be a collection of some $k+1$ of these paths. We will now obtain a collection of $Q_1, \ldots Q_{k+1}$ vertex disjoint paths where the length of each Q_i is at most $2\alpha+1$. To this end, we define each Q_i as some shortest (u,v)-path using the vertices of $V(Q'_i)$. We first claim that the length of Q_i is at most $2\alpha+1$. For the sake of contradiction, suppose not. Then, from Observation 3.3.1, there exist $x, y \in V(Q_i)$ such that $(x,y) \in E(D)$. Since D is acyclic, x appears before y in the path Q_i. This contradicts that Q_i is a shortest (u,v)-path in $V(Q'_i)$. Let $\mathcal{Z} = \bigcup_{i\in[k+1]} V(Q_i)$. Clearly, $\{u,v\} \subseteq \mathcal{Z}$ and $|\mathcal{Z}| \leq (2\alpha-1)(k+1)+2$. The size bound follows because the length of each Q_i is at most $2\alpha+1$, and u, v are the vertices common in each Q_i. To show that $\mathcal{Z}$ is a k-cut preserving set for (u,v) in D, consider the semi-$\mathcal{Z}$-based partition of P that is P itself. Then, $\{Q_1, \ldots, Q_{k+1}\}$ is the list for P containing $k+1$ internally vertex-disjoint $(V(D) \setminus \mathcal{Z})$-free (u,v)-paths. □

3.4.2 Finding a Small k-Cut Preserving Set for a Pair in a Tournament

As explained before, the proof of Lemma 3.4.2 will use induction on α. The next lemma handles the base case where $\alpha = 1$. It is somewhat more complicated com-

pared to the arguments in Section 3.2; the reason for the complication is that we consider the digraph D such that the $D - \{u, v\} \in \mathcal{D}_\alpha$. Thus D is not "exactly" a tournament. This is required in the inductive case for the proof of Lemma 3.4.2.

Lemma 3.4.6. *Let D be an acyclic digraph. Let $u, v \in V(D)$ be such that $N^+(u) = N^-(v) = \emptyset$ and $D - \{u, v\}$ is a tournament. Then, a k-cut preserving set for (u, v) in D of size at most $k^3 + 5k^2 + 3k$ exists and is computable in polynomial time.*

Proof. If all (u, v)-vertex cuts in D have size at least $k + 1$, then the correctness follows from Lemma 3.4.5. Thus, for the rest of the proof assume that there is a (u, v)-vertex-cut in D of size at most k. Let $C = \{c_1, \ldots, c_\ell\}$ be a minimal (u, v)-vertex cut in D of size $\ell \leq k$.

Claim 3.4.3. $C \subseteq N_D^+(u) \cup N_D^-(v)$.

Proof. Suppose not. Then, there exists $c_i \in C$ such that $c_i \notin N^+(u) \cup N^-(v)$. Since C is a minimal (u, v)-vertex cut in D, there exists a path, say P, from u to v in $D - (C \setminus \{c_i\})$. Let u' be the first vertex on P after u and v' be the last vertex of P before v. Since $D - \{u, v\}$ is an acyclic tournament, $(u', v') \in E(D)$. Since $u', v' \notin C$, we get a (u, v)-path in $D - C$, contradicting that C is a (u, v)-vertex cut in D. □

Let $I = \{i \in [\ell] \mid c_i \in N_D^-(v)\}$ and $J = \{j \in [\ell] \mid c_j \in N_D^+(u)\}$. For all $i \in I$, let $U_i = ver_D(u, c_i; C)$ and $D_i = D[U_i]$. For all $j \in J$, let $V_j = ver_D(c_j, v; C)$ and $D_j = D[V_j]$. For all $(i, j) \in [\ell]^2$, $i \neq j$, let $Q_{i,j} = ver_D(c_i, c_j; \emptyset)$ and $D_{i,j} = D[Q_{i,j}]$.

For each $i \in I$ (resp. $j \in J$, resp. $(i, j) \in [\ell]^2$, $i \neq j$), we will compute a k-cut preserving set $\mathcal{Z}_i$ (resp. $\mathcal{Z}_j$, resp. $\mathcal{Z}_{i,j}$) of (u, c_i) (resp. (c_j, v), resp. (c_i, c_j)) in D_i (resp. D_j, resp. $D_{i,j}$) of size at most $2k+3$ (resp. $2k+3$, resp. $k+3$). The procedure to do so is as follows.

- **Computing $\mathcal{Z}_i$, $i \in I$:** First observe that U_i is a candidate for $\mathcal{Z}_i$. Thus, if $|U_i| \leq 2(k + 1)$, set $\mathcal{Z}_i = U_i$. Otherwise, we have that $|U_i| \geq 2k + 3$. Since $D - \{u, v\}$ is an acyclic tournament, let π be the unique topological ordering of $D - \{u, v\}$. We divide this case further into two cases.

  Case 1: $|N^+(u) \cap U_i| \leq k$**:** Let $\widetilde{U}_i$ be the last $k+1$ vertices of U_i in π. Observe that $\widetilde{U}_i \subseteq N^-(c_i) \cap U_i$. Define $\mathcal{Z}_i = (N^+(u) \cap U_i) \cup \widetilde{U}_i \cup \{u, c_i\}$. Clearly, $|\mathcal{Z}_i| \leq 2k + 3$. To prove that $\mathcal{Z}_i$ is a k-cut preserving set for (u, c_i) in

D_i, consider some (u, c_i)-path P in D_i, such that $V(P) \not\subseteq \mathcal{Z}_i$. We will show a $\mathcal{Z}_i$-replacement witness for P in D_i. Consider the semi-$\mathcal{Z}_i$-based partition of P, $P = P_1 \uplus P_2$, where P_1 is the arc $(u, x) \in E(P)$, for some $x \in N^+(u) \cap U_i$ and P_2 is the (x, c_i)-subpath of P. Clearly, $V(P_1) \subseteq \mathcal{Z}_i$. We claim that there are $k+1$ vertex-disjoint (x, c_i)-paths in $\mathcal{Z}_i$. To see this, consider the following argument. Since $V(P) \not\subseteq \mathcal{Z}_i$, there exists a vertex $y \in V(P)$ such that $y \notin \mathcal{Z}_i$. Then, $y \in V(P_2)$. Since $y \notin \mathcal{Z}_i$, it in particular holds that $y \notin \widetilde{U}_i$. Thus, all the vertices of $\widetilde{U}_i$ appear after y in π. Since there is a (x, y)-path in D_i, x appears before y in π. Thus, x appears before all the vertices of $\widetilde{U}_i$ in π. Thus, because $D - \{u, v\}$ is a tournament, $\widetilde{U}_i \subseteq N^+(x) \cap U_i$. Since $\widetilde{U}_i \subseteq N^-(c_i) \cap U_i$, there are $|\widetilde{U}_i|$ many vertex disjoint (x, c_i)-paths in $\mathcal{Z}$. This completes the proof.

Case 2: $|N^+(u) \cap U_i| > k$**:** First observe that all the vertices of $N^+(u) \cap U_i$ appear before c_i in π. Since π is a topological ordering of $D - \{u, v\}$, there are $|N^+(u) \cap U_i| > k$ vertex-disjoint (u, c_i)-paths in $\mathcal{Z}_i$. Thus, each (u, c_i)-vertex-cut in D_i has size at least $k+1$. In this case, let $\mathcal{Z}_i$ be the k-cut preserving set for (u, c_i) in D_i obtained from Lemma 3.4.5. Observe that $|\mathcal{Z}_i| \leq k+3$.

- **Computing $\mathcal{Z}_j$, $j \in J$:** $\mathcal{Z}_j$ can be computed using arguments symmetric to the previous case.

- **Computing $\mathcal{Z}_{i,j}$, $(i, j) \in [\ell]^2$, $i \neq j$:** First observe that all the vertices of $Q_{i,j} \setminus \{c_i, c_j\}$ appear after c_i and before c_j in π. Thus, there are $|Q_{i,j} \setminus \{c_i, c_j\}|$ many vertex-disjoint (c_i, c_j)-paths in $D_{i,j}$. If $|Q_{i,j}| \leq k-2$, then set $\mathcal{Z}_i = Q_{i,j}$, otherwise let $\mathcal{Z}_i$ be the k-cut preserving set for (c_i, c_j) in $D_{i,j}$ obtained from Lemma 3.4.5. In either case, $|\mathcal{Z}_i| \leq k+3$.

Let $\mathcal{Z} := \bigcup_{i \in I} \mathcal{Z}_i \cup \bigcup_{j \in J} \mathcal{Z}_j \cup \bigcup_{(i,j) \in [\ell]^2, i \neq j} \mathcal{Z}_{i,j}$. Observe that $C \subseteq \mathcal{Z}$. First note that $|\mathcal{Z}| \leq |I|(2k+3) + |J|(2k+3) + \ell^2(k+3) \leq k^3 + 5k^2 + 3k^2$ (the last inequality holds because $|I| + |J| = \ell$ and $\ell \leq k$). We will now show that $\mathcal{Z}$ is a k-cut preserving set for (u, v) in D. To see this, consider some(u, v)-path P, in D. Since C is a (u, v)-vertex-cut in D there exists a vertex of C on P. Let c_i be the first vertex of C on P and c_j be the last vertex of C on P (c_i could be the same as c_j). Let P_1 be the (u, c_i)-subpath of P, P_2 be the (c_i, c_j)-subpath of P and P_3 be the (c_j, v)-subpath of P (if c_i is the same as c_j, then P_2 is empty). Thus, $P = P_1 \circ P_2 \circ P_3$ is a semi-$\mathcal{Z}$-based partition of P (as $C \subseteq \mathcal{Z}$). Since $\mathcal{Z}_i$ is a k-cut preserving set for (u, c_i) in D_i, $\mathcal{Z}_i$ is a k-cut preserving set for (c_j, v) in D_j and $\mathcal{Z}_{i,j}$ is a k-cut preserving set for (c_i, c_j)

in $D_{i,j}$, and $\mathcal{Z}_i, \mathcal{Z}_j, \mathcal{Z}_{i,j} \subseteq \mathcal{Z}$, from Observation 3.4.1, $\mathcal{Z}$ is a k-cut preserving set for (u, v) in D. □

3.4.3 Finding a small k-Cut preserving set for a pair in a $D \in \mathcal{D}_\alpha$

We are now ready to prove Lemma 3.4.2.

Lemma 3.4.2 (k-Cut Preserving Lemma)**.** *Let D be an acyclic digraph, and $u, v \in V(D)$ be such that $N^-(u) = N^+(v) = \emptyset$. Additionally, let $D - \{u, v\} \in \mathcal{D}_\alpha$. Then there exists a k-cut preserving set for (u, v) in D of size at most $f(\alpha)$, where $f(1) = k^3 + 5k^2 + 3k$ and for $\alpha > 1$, $f(\alpha) = k^2 g(\alpha) + 2kh(\alpha)$, $g(\alpha) = (2k + (k + kf(\alpha - 1))^2) f(\alpha - 1)$ and $h(\alpha) = (k^2 + k)g(\alpha) + kf(\alpha - 1)$. Moreover, such a set can be found in time $n^{\mathcal{O}(1)}$, where $n = |V(D)|$.*

Proof. We prove this lemma using induction on α. When $\alpha = 1$, the proof follows from Lemma 3.4.6.

Claim 3.4.4. *Let $x, y \in V(D) \setminus \{u, v\}$. Then, a k-cut preserving set for (x, y) of size $g(\alpha)$ in any digraph D' that is a subgraph of D where $u, v \notin V(D')$, can be found in polynomial time.*

Proof. Let W be a minimum (x, y)-vertex-cut in D'. If $|W| > k$, then the claim follows from Lemma 3.4.5. Thus, we are now in the case where $|W| \leq k$. For each $w \in W$, let $\mathcal{Z}(x, w)$ (resp. $\mathcal{Z}(w, y)$) be a k-cut preserving set for (x, w) (resp. (w, y)) in $D'[ver_{D'}(x, w; W)]$ (resp. $D'[ver_{D'}(w, y; W)]$). Let $B = N^-_{D'}(y) \cap \bigcup_{w \in W} \mathcal{Z}(w, y)$. For each $(\mathtt{a}, \mathtt{b}) \in (W \cup B)^2$, let $\mathcal{Z}(\mathtt{a}, \mathtt{b})$ be a k-cut preserving set for $(\mathtt{a}, \mathtt{b})$ in $D'[ver_{D'}(\mathtt{a}, \mathtt{b}; W \cup N^-(y))]$. Then, from Lemma 3.4.4, $\mathcal{Z}(x, y) := \bigcup_{w \in W}(\mathcal{Z}(x, w) \cup \mathcal{Z}(w, y)) \cup \bigcup_{(\mathtt{a},\mathtt{b}) \in (W \cup B)^2} \mathcal{Z}(\mathtt{a}, \mathtt{b})$ is a k-cut preserving set for (x, y) in D'.

We will now show that for any $w \in W$ and $(\mathtt{a}, \mathtt{b}) \in (W \cup B)^2$, each digraph among $D'[ver_{D'}(x, w; W)]$, $D'[ver_{D'}(w, y; W)]$ and $D'[ver_{D'}(\mathtt{a}, \mathtt{b}; W \cup N^-_{D'}(y))]$ has independence number strictly smaller than α. Then, from induction hypothesis and the expression for $\mathcal{Z}(x, y)$ written above, we will conclude that a k-cut preserving set for (x, y) in D' of size $g(\alpha)$ can be found in polynomial time. To see that the independence number of $D'[ver_{D'}(x, w; W)]$ is strictly less than α, observe that y is not adjacent to any vertex in $ver_{D'}(x, w; W)$, as W is an (x, y)-vertex cut in D'. Thus, any independent set of $D'[ver_{D'}(x, w; W)]$ together with y is an

independent set of D' and hence of D. Since $y \notin \{u, v\}$, $u, v \notin V(D')$ and the independence number of $D - \{u, v\}$ is α, we have that the independence number of $D'[ver_{D'}(x, w; W)]$ is strictly smaller than α. A similar argument holds for $D'[ver_{D'}(w, y; W)]$ as in this case x is not adjacent to any vertex of $ver_{D'}(w, y; W)$. For $D'[ver_{D'}(\mathtt{a}, \mathtt{b}; W \cup N^-_{D'}(y))]$, since $ver_{D'}(\mathtt{a}, \mathtt{b}; W \cup N^-_{D'}(y)) \cap N^-_{D'}(y) = \emptyset$, $u, v \notin V(D')$ and $N^+_{D'}(y) = \emptyset$, any independent set of $D'[ver_{D'}(\mathtt{a}, \mathtt{b}; W \cup N^-_{D'}(y))]$ together with y is an independent set in $D - \{x, y\}$. Since $D - \{x, y\}$ has independence number α, $D'[ver_{D'}(\mathtt{a}, \mathtt{b}; W \cup N^-_{D'}(y))]$ has independence number strictly smaller than α. □

Let C be a minimum (u, v)-vertex-cut in D. If $|C| > k$, then the lemma follows from Lemma 3.4.5. Thus, for the remainder of the proof we assume that $|C| \leq k$. For each $c \in C$, let $U_c = ver_D(u, c; C)$, $V_c = ver_D(c, v; C)$, $\mathcal{Z}(u, c)$ be a (u, c) k-cut preserving set in $D[U_c]$, and $\mathcal{Z}(c, v)$ be a (c, v) k-cut preserving set in $D[V_c]$. For each $(c, c') \in C^2$, $c \neq c'$, let $Q_{c,c'} = ver_D(c, c'; C)$, and $\mathcal{Z}(c, c')$ be a k-cut preserving set in $D[Q_{c,c'}]$. Then from Lemma 3.4.3, $\mathcal{Z} := \bigcup_{c \in C} \mathcal{Z}(u, c) \cup \mathcal{Z}(c, v) \cup \bigcup_{(c,c') \in C^2, c \neq c'} \mathcal{Z}(c, c')$ is a k-cut preserving set for (u, v) in D. Since $C \cap \{u, v\} = \emptyset$, from Claim 3.4.4, for each $(c, c') \in C^2$, $c \neq c'$, $\mathcal{Z}(c, c')$ of size $g(\alpha)$ can be computed in polynomial time. In the remainder of the proof, we will show how to compute $\mathcal{Z}(u, c)$ and $\mathcal{Z}(c, v)$, for any $c \in C$, of the desired size. We will only give the proof of construction of $\mathcal{Z}(u, c)$ as the proof for $\mathcal{Z}(c, v)$ is symmetrical.

Claim 3.4.5. *For any $c \in C$, $\mathcal{Z}(u, c)$ of size $h(\alpha)$ can be computed in polynomial time.*

Proof. For ease of notation, let $\widehat{D} = D[U_c]$. Let A be a minimum (u, c)-vertex-cut in $\widehat{D}$. First note that $A \cap \{u, v\} = \emptyset$. If $|A| > k$, then the claim follows from Lemma 3.4.5. Thus, for the remainder of the proof, assume that $|A| \leq k$.

For each $a \in A$, let $\widehat{U_a} = ver_{\widehat{D}}(u, a; A)$, $\widehat{V_a} = ver_{\widehat{D}}(a, c; A)$, $\widehat{\mathcal{Z}}(u, a)$ be a (u, a) k-cut preserving set in $\widehat{D}[\widehat{U_a}]$ and $\widehat{\mathcal{Z}}(a, c)$ be a (a, c) k-cut preserving set in $\widehat{D}[\widehat{V_a}]$. For each $(a, a') \in A^2$, $a \neq a'$, let $R_{a,a'} = ver_{\widehat{D}}(a, a'; A)$ and $\widehat{\mathcal{Z}}(a, a')$ be a k-cut preserving set in $\widehat{D}[R_{a,a'}]$. Then from Lemma 3.4.3, $\mathcal{Z}(u, c) := \bigcup_{a \in A}(\widehat{\mathcal{Z}}(u, a) \cup \widehat{\mathcal{Z}}(a, c)) \cup \bigcup_{(a,a') \in A^2, a \neq a'} \widehat{\mathcal{Z}}(a, a')$ is a k-cut preserving set for (u, c) in D. Since $A \cap \{u, v\} = \emptyset$ and $c \in \{u, v\}$, from Claim 3.4.4, for each $a \in A$, $(a, a') \in A^2$, $a \neq a'$, $\widehat{\mathcal{Z}}(a, c)$ and $\widehat{\mathcal{Z}}(a, a')$ of size $g(\alpha)$ can be computed in polynomial time. Moreover, the independence number of $\widehat{D}[\widehat{U_a}] - \{u, a\}$ is strictly smaller than α because $c (\neq v)$ is not adjacent to any vertex in $\widehat{U_a}$, besides possibly u and a. Thus, for each $a \in A$,

a set $\widehat{\mathcal{Z}}(u,a)$ of size $f(\alpha-1)$ can be computed in polynomial time by the induction hypothesis. This finishes the proof of the claim. □

Thus, from the previous arguments and Claim 3.4.5, we have that $\mathcal{Z}$ is a k-cut preserving set for (u,v) in D of size at most $k^2g(\alpha)+2kh(\alpha)$. □

A rough computation gives that, for any k, $g(\alpha) \leq 6k^2 f(\alpha-1)$ and $h(\alpha) \leq 8k^4 f(\alpha-1)$. This imply that $f(\alpha) \leq 22k^5 f(\alpha-1)^3$. By noting that $f(1) \leq 22k^5$, we can show the following observation.

Observation 3.4.1. *For any α and k, there exists a k-cut preserving set of size smaller than $f(k,\alpha) = (22k^5)^{4^\alpha}$.*

3.4.4 k-Cut Preserving Sets for a Set of Vertices

Below we also define a notion of k-cut preserving sets for a set of vertices. Such a notion will come handy in our applications. Given a digraph D and $X \subseteq V(D)$, for each $(u,v) \in X^2$, we define the digraph $D^X_{(u,v)}$ as follows (note that u could be equal to v). Let $R = V(D) - X$. Then, $D^X_{(u,v)}$ is the supergraph of $D[R]$ obtained by adding two new vertices u^+ and v^- together with the following set of additional arcs: $\{(u^+,x) : x \in R, (u,x) \in E(D)\} \cup \{(x,v^-) : x \in R, (x,v) \in E(D)\}$.

Definition 3.4.2 (k-Cut Preserving Set for a Set of Vertices)**.** *For any digraph D, a positive integer k and $X \subseteq V(D)$, we say that $X \subseteq \mathcal{Z} \subseteq V(D)$ is a* k-cut preserving set for X*, if for all $(u,v) \in X^2$, $\mathcal{Z}$ is a k-cut preserving set for (u,v) in $D^X_{(u,v)}$.*

Lemma 3.4.7. *For any digraph $D \in \mathcal{D}_\alpha$, a positive integer k, and $S \subseteq V(D)$ such that $D-S$ is a acyclic, a k-cut preserving set for S of size at most $|S|^2 f(k,\alpha)$ can be found in polynomial time, where $f(k,\alpha) \leq (22k^5)^{4^\alpha}$.*

Proof. For each pair $(u,v) \in S^2$ (u and v could be equal), let $\mathcal{Z}_{(u,v)}$ be the a k-cut preserving set for (u^+,v^-) in $D^S_{(u,v)}$ obtained from Lemma 3.4.2. From the definition of k-cut preserving set for S, $\mathcal{Z} = \bigcup_{(u,v)\in S^2} \mathcal{Z}_{(u,v)}$ is a k-cut preserving set for S. From Observation 3.4.1, for any $(u,v) \in S^2$, $|\mathcal{Z}_{(u,v)}| \leq f(k,\alpha)$. Thus, we conclude the correctness of the lemma. □

3.5 Applications of the k-Cut Preserving Lemma

3.5.1 Fault-Tolerant (S, S)-Reachability

In this section, we prove Lemma 3.1.1. Recall that (D, S, ℓ, k) is an instance of FTR(S, S) where $D \in \mathcal{D}_\alpha$, $S \subseteq V(D)$ and ℓ, k are positive integers such that each strongly connected component of $D - S$ has size at most ℓ. The goal is to compute a subgraph H of D of size $k^{2^{\mathcal{O}(\alpha)}}$ such that, for any $A \subseteq E(D)$ of size at most k, for any $s, t \in S$, if $D - A$ has an (s, t)-path, then so does $H - A$. It is not difficult to see from Lemma 3.4.1 that if $\mathcal{Z}$ is a k-cut preserving set for S in D, then $H = D[\mathcal{Z}]$ is a solution for (D, S, ℓ, k) (for any ℓ). When $\ell = 1$, $D - S$ is acyclic and hence a k-cut preserving set for S can be computed using Lemma 3.4.7. When $\ell > 1$, in order to use Lemma 3.4.7 we modify the digraph D to turn $D - S$ acyclic. We now describe the operation, which we call dagify, that is used to turn $D - S$ acyclic. Informally, for each strongly connected component SC of D we turn it into an independent set while preserving the paths in D that use the vertices of SC. This is achieved by creating a new vertex for every ordered pair of vertices (say, (u, v)) in SC. Such a vertex represents the existence of a (u, v)-path in the strongly connected component SC. In fact, in the path in the modified graph, each new vertex corresponding to some pair (u, v) can be replaced by some (u, v)-path from the strongly connected component SC to yield a path in the original graph. Then, arcs between two vertices in this newly constructed vertex set are put in such a way that the concatenation of the paths corresponding to these new vertices gives a path in D. This idea is formalized below.

Definition 3.5.1 (dagify(D, R))**.** *Let D be a digraph, $R \subseteq V(D)$ and $S = V(D) \setminus R$. Let $SC_1, \ldots, SC_d$ be the strongly connected components of $D[R]$. For $a \in [d]$, let $V(SC_a) = \{v_1^a, \ldots, v_{n_a}^a\}$, where $n_a = |V(SC_a)|$. Then, $D_R^\dagger :=$ dagify(D, R) is the digraph defined as:*

Vertex set of $D_R^\dagger$: *For each $a \in [d]$, let $SC_a^\dagger = \{\mathbf{v}_{ij}^a \mid (v_i^a, v_j^a) \in \{SC_a\}^2, i, j \in [n_a]\}$. Let $R^\dagger = \cup_{a \in [d]} SC_a^\dagger$ and $V(D_R^\dagger) = R^\dagger \cup S$.*

Arc set of $D_R^\dagger$: *It contains all the arcs of D with both end-points in S. For each $a \in [d]$, $SC^\dagger{}_a$ is an independent set in $D_R^\dagger$. For any $a \in [d]$, $s \in S$ and $i, j \in [n_a]$, $(s, \mathbf{v}_{ij}^a) \in E(D_R^\dagger)$ if and only if $(s, v_i^a) \in E(D)$. Similarly, $(\mathbf{v}_{ij}^a, s) \in E(D_R^\dagger)$ if and only if $(v_j^a, s) \in E(D)$. We put the arcs between $SC_a^\dagger$ and $SC_b^\dagger$, for distinct $a, b \in [d]$ as follows. For any $i, j \in [n_a]$ and $i', j' \in [n_b]$, $(\mathbf{v}_{ij}^a, \mathbf{v}_{i'j'}^b) \in E(D_R^\dagger)$ if and only if $(v_j^a, v_{i'}^b) \in E(D)$.*

For a set of vertices of $X^\dagger \subseteq D_R^\dagger$, $full\text{-}comp(X^\dagger)$ denotes the set of vertices of $V(D)$ such that, for each $\mathbf{v}_{i,j}^a \in X^\dagger$, all the vertices of SC_a belong to full-comp($X^\dagger$). Also all the vertices of S that belong to $X^\dagger$, belong to $full\text{-}comp(X^\dagger)$. Observe that $|full\text{-}comp(X^\dagger)| \leq \ell^2 \cdot |X^\dagger|$, where ℓ is the upper bound on the size of each SC_a. Note from the construction above that, for any $s, t \in S$ and an (s,t)-path $P^\dagger$ in $D_R^\dagger$, there exists an (s,t)-path P in D such that $V(P) \subseteq full\text{-}comp(P^\dagger)$. The following observations state a few properties of the digraph $D_R^\dagger$ that would be useful when we want to find a k-cut preserving set for $D_R^\dagger$ using Lemma 3.4.7.

Observation 3.5.1. *$D_R^\dagger[R^\dagger]$ is acyclic.*

Proof. Recall, from the construction of $D_R^\dagger$, that $R^\dagger = \bigcup_{a\in[d]} SC_a^\dagger$ and each $SC_a^\dagger$ is an independent set in $D_R^\dagger$. Without loss of generality, let $SC_1, \ldots, SC_d$ be the strongly connected components of $D[R]$ ordered as in their topological ordering. Then, there is no arc from a vertex of SC_b to a vertex of SC_a, for any $b > a$, in D. Thus, from the construction of $D_R^\dagger$, there is no arc from any $\mathbf{v}_{ij}^b$ to any $\mathbf{v}_{i'j'}^a$ $(b > a)$. This shows that $D_R^\dagger[R^\dagger]$ is acyclic. □

Observation 3.5.2. *If $D \in \mathcal{D}_\alpha$ and every strongly connected component of $D[R]$ has size at most ℓ, then $D_R^\dagger \in \mathcal{D}_{\ell^2\alpha}$.*

Proof. Recall that $R^\dagger = \bigcup_{a\in[d]} SC^\dagger{}_a$ and $D_R^\dagger[SC_a^\dagger]$ has no arc. From the construction of $D_R^\dagger$, for each $a \in [d]$, $|SC_a^\dagger| \leq \ell^2$. Finally, since $D \in \mathcal{D}_\alpha$, from the construction of $D_R^\dagger$, the size of any maximum independent set in $D_R^\dagger$ is at most $\max_{a\in[d]} |SC_a^\dagger| \cdot \alpha \leq \ell^2\alpha$. □

We define some terminology that would come handy later. For any $A \subseteq E(D)$, we say that a vertex $v \in V(D)$ *is affected by* A if there exists some arc of A that is incident on v. The *set affected by A in $D_R^\dagger$* is the set of vertices of $D_R^\dagger$ containing the union of the vertices in $SC_a^\dagger$, for each $a \in [d]$ such that a vertex in SC_a is affected by A in D.

Observation 3.5.3. *Let D be a digraph, $R \subseteq V(D)$ and $S = V(D) \setminus R$. Let $A \subseteq E(D)$ of size at most k. Let $\mathcal{A}^\dagger$ be the set affected by A in $D_R^\dagger$. Recall the construction of $D_R^\dagger$ from Definition 3.5.1. For some $\mathbf{v}_{ij}^a, \mathbf{v}_{i'j'}^b \in R^\dagger$, let $P^\dagger$ be an $\mathcal{A}^\dagger$-free $(\mathbf{v}_{ij}^a, \mathbf{v}_{i'j'}^b)$-path in $D_R^\dagger$. Then there exists a $(v_i^a, v_{j'}^b)$-path P in D such that: $V(P) \subseteq full\text{-}comp(P^\dagger)$ and, P does not use any arc of A.*

Proof. Recall the construction of $\mathsf{dagify}(D, R)$. Consider any path P obtained from $P^\dagger$ by replacing all the vertices of $R^\dagger$ as follows. If for any $c \in [d]$, $i^*, j^* \in [n_c]$, $\mathbf{v}_{i^*j^*}^c \in$

$V(P^\dagger)$, then replace $\mathbf{v}^c_{i^*j^*}$ in $P^\dagger$ by any $(v^c_{i^*}, v^c_{j^*})$-path in the strongly connected component SC_c. Clearly, the path P obtained is a $(v^a_i, v^b_{j'})$-path in D and $V(P) \subseteq$ *full-comp*$(P^\dagger)$. Also from the definition of $\mathcal{A}^\dagger$ and the fact that $P^\dagger$ is $\mathcal{A}^\dagger$-free, we get that P cannot use an arc of A. □

From the construction in Definition 3.5.1, for any $s, t \in S$, for an (s,t)-path P in D, we can associate a unique (s,t)-path $P^\dagger$ in $D^\dagger_R$. This is elaborated below. Consider the digraph $D^\dagger_R$ obtained by $\mathsf{dagify}(D, R)$. $(v^a_i, v^a_j) \in SC^2_a$ for some component SC_a of $D[R]$. Let $s, t \in S$. Let P be an S-free (s,t)-path in D. For any such path P, we define the notion of a *reduced path* of P in $D^\dagger_R$ as follows. Consider the unique partition $P = P_s \circ P_{i_1} \circ \ldots \circ P_{i_q} \circ P_t$ such that P_s is an arc (s,u) where $u \in V(SC_{i_1})$, P_t is an arc (v,t) where $v \in V(SC_{i_q})$ and for each $j \in [q]$, $V(P_{i_j}) \subseteq V(SC_{i_j})$, where $i_1, \ldots, i_j \in [d]$ and $i_1 < \ldots < i_q$. For each $j \in [q]$, let P_{i_j} be a $(v^{i_j}_{p_j}, v^{i_j}_{r_j})$-path. Consider the vertex $\mathbf{v}^{i_j}_{p_j,r_j}$ in $V_{i_j} \subseteq R^\dagger \subseteq V(D^\dagger_R)$. From the construction of $D^\dagger_R$, we get the (s,t)-path $P^\dagger = s \circ \mathbf{v}^{i_1}_{p_1,r_1} \circ \mathbf{v}^{i_2}_{p_2,r_2} \circ \ldots \circ \mathbf{v}^{i_q}_{p_q,r_q} \circ t$ in $D^\dagger_R$. This (s,t)-path $P^\dagger$ in $D^\dagger_R$ is called the *reduced path of* P *in* $D^\dagger_R$.

Proof of Lemma 3.1.1 Recall (D, S, ℓ, k) is an instance of FTR(S,S). Let $R = V(D) \setminus S$. Let $D^\dagger_R$ be obtained by $\mathsf{dagify}(D, R)$. From Observations 3.5.1 and 3.5.2, Lemma 3.4.7 can be used to compute a $(2k\ell^2+1)$-cut preserving set for S in $D^\dagger_R$. Let $\mathcal{Z}^\dagger$ be such a set. Let $\mathcal{Z} = $ *full-comp*$(\mathcal{Z}^\dagger)$. We claim that $H = D[\mathcal{Z}]$ is a solution to the instance (D, S, ℓ, k). (First note that the size bound on H follows from Lemma 3.4.7 and the fact that each strongly connected component of R has size at most ℓ.)

Towards this let $A \subseteq E(D)$ of size at most k, $s, t \in S$ and P be an (s,t)-path in $D - A$. We need to show that there is some (s,t)-path in $H - A$ too. Let $P = P_1 \circ \ldots \circ P_q$ be the S-based partition of P such that each P_i is an (s_i, t_i)-path. Then it suffices to show that for each fixed $i \in [q]$, there is some (s_i, t_i) path in $H - A$ (these paths would yield a closed walk from s to t in $H - A$ and hence an (s,t)-path in $H - A$). In the remaining part of the proof, we focus on proving this. Note that each P_i is S-free. Fix any $i \in [q]$. For the ease of notation, let us call the path P_i as P, vertices s_i, t_i as s, t respectively.

Let $P^\dagger$ be the reduced path corresponding of P in $D^\dagger_R$. Since $\mathcal{Z}^\dagger$ is a $(2k\ell^2+1)$-cut preserving set for $P^\dagger$ in $D^\dagger_R$, consider a $\mathcal{Z}^\dagger$-witnessing replacement $P^\dagger = P^\dagger_1 \circ \ldots \circ P^\dagger_r$. Recall the notation from the construction in Definition 3.5.1.

For an arbitrary $c \in [r]$, let $P^\dagger_c$ be a $(\mathbf{v}^a_{ij}, \mathbf{v}^b_{i',j'})$-path (or $(s, \mathbf{v}^a_{ij})$-path or $(\mathbf{v}^a_{ij}, s)$-path). Observe that, since $P^\dagger$ is the reduced path of P, to finish the proof of the

lemma, it is enough to show a (v_i^a, v_j^b)-path (or (s, v_i^a)-path or (v_i^a, s)-path) exists in $H - A$. Without loss of generality, let $P_c^\dagger$ be a $(\mathbf{v}_{ij}^a, \mathbf{v}_{i',j'}^b)$-path, the other cases hold due to similar arguments.

As $P^\dagger = P_1^\dagger \circ \ldots \circ P_d^\dagger$ is a $\mathcal{Z}^\dagger$-witnessing replacement, one of the following cases arises.

1. $V(P_c^\dagger) \subseteq \mathcal{Z}^\dagger$. Since $P^\dagger$ is the reduced path of P, consider the (v_i^a, v_j^b)-subpath, say P_c', of P. Then, $V(P_c') \subseteq \mathit{full\text{-}comp}(P_c^\dagger) \subseteq \mathcal{Z}$ (because $V(P_c^\dagger) \subseteq \mathcal{Z}^\dagger$). Also since P does not have an arc in A, so does P_c'. Thus, by the construction of H, P_c' is a path in $H - A$.

2. There is a list $\mathcal{L}_i$ of $2k\ell^2 + 1$ internally vertex-disjoint $(\mathbf{v}_{ij}^a, \mathbf{v}_{i'j'}^b)$-paths in $D_R^\dagger[\mathcal{Z}^\dagger]$. Let $\mathcal{A}^\dagger$ be the set of affected vertices of A in $D_R^\dagger$. Clearly, $|\mathcal{A}^\dagger| \leq 2k\ell^2$. Then there exists a path in $\mathcal{L}_i$ that is $\mathcal{A}^\dagger$-free. Then from Observation 3.5.3, there exists a (v_i^a, v_j^b)-path, say P_c', such that $V(P_C') \subseteq \mathit{full\text{-}comp}(P_c^\dagger) \subseteq \mathcal{Z}$ and, that does not use an arc of A. From the construction of H, P_c' is a path in $H - A$.

This finishes the proof of the lemma.

3.5.2 Kernel for DFAS on $\mathcal{D}_\alpha$

In this section, we give a polynomial kernel for DFAS on $\mathcal{D}_\alpha$, that is, we prove Theorem 3.1.1.

Theorem 3.1.1. DFAS *on* $\mathcal{D}_\alpha$ *admits a kernel of size* $k^{\mathcal{O}(4^\alpha)}$.

We achieve this in two steps. In the first step, we find a set of vertices S of size $\mathcal{O}(\alpha k)$ whose removal results in an acyclic digraph. We then show that it is enough to keep the vertices of a k-cut-preserving set for S to get a kernel.

Lemma 3.5.1. *Let* (D, k) *be an instance of* DFAS *and let* $D \in \mathcal{D}_\alpha$. *In polynomial time, one can either correctly conclude that* (D, k) *is a* NO *instance of* DFAS, *or output a set* $S \subseteq V(D)$ *such that* $|S| \leq (2\alpha + 1)k$ *and* $D - S$ *is acyclic.*

Proof. Since $D \in \mathcal{D}_\alpha$, if there exists a cycle in D, then from Observation 3.3.1 there exists a cycle of length at most $2\alpha + 1$. Thus, one can greedily find vertex-disjoint cycles each of length at most $2\alpha+1$. To see this, notice that after the removal of any

vertex set, the resulting digraph remains in $\mathcal{D}_\alpha$ and hence our previous argument reapplies. If one finds more than k such cycles, then any *dfas* of S has size at least $k+1$, in which case report that (D,k) is a NO instance. Otherwise, one finds a collection $\mathcal{C}$ of at most k cycles of length at most $2\alpha+1$ each such that every cycle of D intersects in some vertex of one of the cycles in $\mathcal{C}$. In this case, output S as the union of the vertex sets of the cycles in $\mathcal{C}$. □

Lemma 3.5.2. *Let (D,k) be an instance of* DFAS *where $D \in \mathcal{D}_\alpha$. Let S be a set computed by Lemma 3.5.1 on input (D,k). Let $\mathcal{Z}$ be a k-cut preserving set for S in D computed by Lemma 3.4.7. Then, (D,k) is a* YES *instance of* DFAS *if and only if $(D[\mathcal{Z}],k)$ is a* YES *instance.*

Proof. Since $D[\mathcal{Z}]$ is a subgraph of D, the forward direction is trivial. For the backward direction, let A be a *dfas* of $D[\mathcal{Z}]$ of size at most k. We will prove that A is also a *dfas* of D. For the sake of contradiction, suppose that A is not a *dfas* of D, that is, there is some cycle C in $D-A$. Since $D-S$ is acyclic, C must contain some vertex from S. Let $v_0,\ldots,v_\ell$ be the vertices in $V(C)\cap S$, appearing in this order along C (the choice of which vertex is denoted v_0 is arbitrary).

For any pair (v_i,v_{i+1}), the (v_i,v_{i+1})-subpath of C is S-free. Recall the construction of the digraph $D^S_{(v_i,v_{i+1})}$ before Definition 3.4.2. From the definition of $\mathcal{Z}$ (by Definition 3.4.2), $\mathcal{Z}$ contains a k-cut preserving set for (v_i,v_{i+1}) in $D^S_{v_i,v_{i+1}}$, for each $i\in[0,\ell]$. From Lemma 3.4.1, there exists a path P_i' from v_i to v_{i+1} in $D[\mathcal{Z}]-A$. Thus, we conclude that for each $i\in[\ell]_0$, there exists a (v_i,v_{i+1})-path (where addition is modulo ℓ) in $D[\mathcal{Z}]-A$. Since C is a cycle, these paths give a closed walk (and hence also a cycle) in $D[\mathcal{Z}]-A$. □

Proof of Theorem 3.1.1: Its correctness follows from Lemmas 3.4.7 and 3.5.2 by noting that the size of the set $\mathcal{Z}$ obtained is smaller than $((2\alpha+1)k)^2 f(k,\alpha)$.

Chapter 4

Kernel for DEOCT on Bounded Independence Number Digraphs via Parity Reachability Preserving Fault-Tolerant Subgraphs

The Directed Edge Odd Cycle Transversal (DEOCT) problem is the parity-based version of DFAS, formally defined as follows. (On general digraphs, the vertex and arc versions of the problem are equivalent [150]).

Directed Edge Odd Cycle Transversal (DEOCT) **Parameter:** k
Input: A digraph D and a non-negative integer k.
Question: Does there exist $S \subseteq E(D)$ of size at most k such that $D - S$ has no odd cycle?

In this chapter, we first prove that DEOCT is NP-hard on tournaments. We later prove that DEOCT admits a polynomial kernel on the class of digraphs of bounded independence number. The later result is obtained by building a connection of this problem with parity reachability preserving fault-tolerant subgraphs.

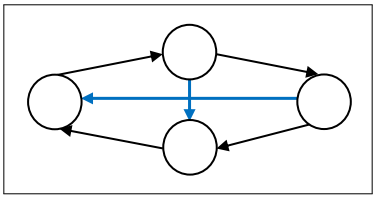


Figure 4.1: A directed edge odd cycle transversal (in blue) that is not a directed feedback arc set.

4.1 NP-hardness of DEOCT on Tournaments

Observe that a tournament has no directed cycle if and only if it has no directed triangle (a cycle on three vertices). In turn, this simple observation implies that, given a tournament D, any subset S of the *vertices* of D has the following property: $D-S$ is a DAG if and only if it has no directed odd cycle. Thus, the vertex versions of DFAS and DEOCT on tournaments are equivalent. However, for DFAS and DEOCT the situation is not so clear. Indeed, it is not difficult to come up with a tournament D and a subset of *arcs* S of D such that $D-S$ is not a DAG, yet it has no directed odd cycle (see, e.g., Fig. 4.1). Nonetheless, we are able to prove that given a tournament D and a subset S of the *arcs* of D such that $D-S$ has no directed odd cycle, there exists a subset of arcs S' of D such that $D-S'$ is a DAG and $|S'| \leq |S|$. In particular, we show that DEOCT on tournaments is equivalent to DFAS on tournaments. We thus establish the following result.

Theorem 4.1.1. DEOCT *on tournaments is* NP*-hard.*

Given a digraph D, observe that any directed feedback arc set solution (*dfas*) of D is also a directed odd cycle transversal (*deoct*) of D. But the converse may not always be true. But what we can prove in the converse case is that if $|S|$ is an *deoct* of D, then there exists a vertex set $|S'|$ such that $|S'| \leq |S|$ and S' is a *dfas* of D. Lemma 4.1.2 proves this. The following lemma from [181] will be used in the proof of Lemma 4.1.2.

Lemma 4.1.1 ([181] Lemma 11)**.** *Let D be a tournament on n vertices and m arcs. Then D has a dfas of size at most $\frac{m}{2} - \frac{1}{2}\lceil\frac{n-1}{2}\rceil$.*

Lemma 4.1.2. *Let D be a tournament. If S is a deoct of D, then there exists a dfas of D of size at most $|S|$.*

Proof. Consider the digraph $D-S$. Let $C_1, \ldots, C_t$ be the set of strongly connected components of $D-S$. For ease of notation, let us denote $D[C_i]$ by D_i. Let $S_i =$

$S \cap E(D_i)$. For each $i \in [t]$, we will now construct a set X_i such that X_i is a *dfas* of D_i such that $|X_i| \leq |S_i|$. Since D_i is a strongly connected digraph and has no odd directed cycles, from Observation 4.4.1, D_i is bipartite. Let (A_i, B_i) be a bipartition of D_i. Let $|A_i| = n_a$ and $|B_i| = n_b$.

Claim 4.1.1. $|S_i| \geq \binom{n_a}{2} + \binom{n_b}{2}$.

Proof. Since (A_i, B_i) is a bipartition of D_i, $D_i[A_i]$ and $D_i[B_i]$ are independent. Since D_i is an induced subgraph of $D - S$ and D is a tournament, all the arcs whose both end-points are either in A_i or in B_i, are in S. Thus, the claim follows. □

Let X_i^a and X_i^b be the *dfas* of $D[A_i]$ and $D[B_i]$ obtained from Lemma 4.1.1. Then $|X_i^a| \leq \frac{1}{2}\binom{n_a}{2} - \frac{1}{2}\lceil\frac{n_a-1}{2}\rceil$ and $|X_i^b| \leq \frac{1}{2}\binom{n_b}{2} - \frac{1}{2}\lceil\frac{n_b-1}{2}\rceil$. Consider $E(A_i, B_i)$ and $E(B_i, A_i)$. If $|E(A_i, B_i)| \leq |E(B_i, A_i)|$, then let $X_i^{ab} = E(A_i, B_i)$. Otherwise, let $X_i^{ab} = E(B_i, A_i)$. Observe that $|X_i^{ab}| \leq \frac{n_a n_b}{2}$. Let $X_i = X_i^a \cup X_i^b \cup X_i^{ab}$.

Claim 4.1.2. *X_i is a dfas of D_i.*

Proof. Suppose not. Then there is a cycle, say Q, in $D_i - X_i$. Recall (A_i, B_i) is a bipartition of D_i. Since X_i^a is a *dfas* of $D[A_i]$ and $X_i^a \subseteq X_i$, Q is not entirely contained in A_i. Similarly, Q is not entirely contained in B_i. Thus, if such a cycle Q exists, it has to intersect both A_i and B_i. This implies there exists two distinct arcs of Q, say e_1 and e_2, such that $e_1 \in E(A_i, B_i)$ and $e_2 \in E(B_i, A_i)$. But this is not possible, because $X_i^{ab} \subseteq X$. □

Claim 4.1.3. $|X_i| \leq |S_i|$.

Proof. From the construction of X_i, we have $|X_i| = |X_i^a| + |X_i^b| + |X_i^{ab}$. Thus,

$$\begin{aligned}
|X_i| &\leq \frac{1}{2}\binom{n_a}{2} - \frac{1}{2}\lceil\frac{n_a-1}{2}\rceil + \frac{1}{2}\binom{n_b}{2} - \frac{1}{2}\lceil\frac{n_b-1}{2}\rceil + \frac{n_a n_b}{2} \\
&= \frac{n_a^2}{4} - \frac{n_a}{4} - \frac{1}{2}\lceil\frac{n_a-1}{2}\rceil + \frac{n_b^2}{4} - \frac{n_b}{4} - \frac{1}{2}\lceil\frac{n_b-1}{2}\rceil + \frac{n_a n_b}{2} \\
&\leq \frac{n_a^2}{4} - \frac{n_a}{4} - \frac{1}{2}(\frac{n_a}{2} - \frac{1}{2}) + \frac{n_b^2}{4} - \frac{n_b}{4} - \frac{1}{2}(\frac{n_b}{2} - \frac{1}{2}) + \frac{n_a n_b}{2}
\end{aligned}$$

Since $n_a n_b \leq \frac{n_a^2+n_b^2}{2}$, we have the following.

$$|X_i| \leq \frac{n_a^2}{2} + \frac{n_b^2}{2} - \frac{n_a}{2} - \frac{n_b}{2} + \frac{1}{8} = \binom{n_a}{2} + \binom{n_b}{2} + \frac{1}{8}$$

Since, $\binom{n_a}{2} + \binom{n_b}{2}$ is an integer and the size of the set X_i is an integer, we have that $|X_i| \leq \binom{n_a}{2} + \binom{n_b}{2} \leq |S|$. The last inequality follows from Claim 4.1.1.

□

Let $S' = S \setminus \cup_{i \in [t]} S_i$. Observe that $S = S' \uplus S_1 \uplus \ldots \uplus S_t$. Let $X = \cup_{i \in [t]} X_i \cup S'$.

Claim 4.1.4. $|X| \leq |S|$.

Proof. Since $X = \cup_{i \in [t]} X_i \cup S'$, $|X| = \cup_{i \in [t]} |X_i| + |S'|$. Thus, from Claim 4.1.3, $|X| \leq \cup_{i \in [t]} |X_i| + |S'| \leq |S|$. □

Claim 4.1.5. *X is a dfas of D.*

Proof. For the sake of contradiction, suppose there is a cycle, say Q, in $D - X$. Recall that $C_1, \ldots, C_t$ are the strongly connected components of $D - S$. Also, S' is the set of those arcs of S whose one endpoint belong to C_i and the other in C_j, for some $i, j \in [t]$, $i \neq j$. Since $S' \subseteq X$, the vertex set of Q cannot intersect both C_i and C_j for some $i, j \in [t]$, $i \neq j$. Thus, the vertex set of Q is fully contained in some C_i. Since $X_i \subseteq X$ and X_i is a *dfas* of D_i (from Claim 4.1.2), there is no cycle in $D_i - X$. This proves the claim. □

Claim 4.1.4 and 4.1.5 prove the lemma. □

Lemma 4.1.3. *Let D be a tournament. For any integer k, D has a dfas of size at most k if and only if D has a deoct of size at most k.*

Proof. Clearly, any *dfas* of D is also a *deoct* of D. The other direction follows from Lemma 4.1.2. □

Since DFAS on tournaments is NP-hard [197], from Lemma 4.1.3 it follows that, DEOCT on tournaments in NP-hard. This proves Theorem 4.1.1.

4.2 Parity Preserving Fault-Tolerant Subgraphs

The question of the parameterized complexity of DEOCT was explicitly stated as an open problem [75] for the first time in 2007, immediately after the announcement of the first parameterized algorithm for DFAS. Since then, the problem has been restated several times [53, 56, 158, 155]. Recently, Lokshtanov et al. [150] proved that DEOCT is W[1]-hard. Specifically, this means that DEOCT is highly unlikely to be FPT or admit a kernel of any size (even exponential in k). Based on the parity-based generalization of our combinatorial lemma (Lemma 4.2.1), we establish a polynomial kernel for DEOCT on $\mathcal{D}_\alpha$, which stands in sharp contrast to its aforementioned status on general digraphs.

Theorem 4.2.1. DEOCT *on* $\mathcal{D}_\alpha$ *admits a kernel of size* $(\alpha k)^{\mathcal{O}(4^{4\alpha^3})}$.

As in Chapter 3, the proof of Theorem 4.2.1 is established via fault-tolerant subgraphs. In this case, we define a parity version of the fault-tolerant subgraph problem. In the Fault-Tolerance (S,T)-Parity Reachability problem, we are given a digraph D, two terminal sets $S, T \subseteq V(D)$, positive integers k and p, and a non-negative integer r. The objective is to construct a subgraph H of D with as few arcs/vertices as possible, such that, after the failure of any set of at most k arcs in D, the following property is preserved for any two vertices $s \in S$ and $t \in T$: if there exists a directed path from s to t in D whose length q satisfies $(q \mod p = r)$, then there also exists a directed path from s to t in H whose length q' satisfies $(q' \mod p = r)$. We prove the following combinatorial lemma which aids the proof of Theorem 4.2.1.

Lemma 4.2.1. *Given a digraph* $D \in \mathcal{D}_\alpha$*, positive integers* k, ℓ, p*, a non-negative integer* r*, and* $S \subseteq V(D)$ *such that every strongly connected component of* $D - S$ *has at most* ℓ *vertices, the* Fault-Tolerance (S,S)-Parity Reachability *problem admits a solution* H *on* $(|S|\alpha\ell pk)^{\mathcal{O}(4^{\alpha\ell^2})}$ *vertices. Furthermore, such a solution* H *can be found in polynomial time.*

In fact, we present combinatorial results stronger than Lemma 4.2.1 that yield a polynomial kernel for a more general version of DEOCT, where instead of hitting directed odd cycles, the objective is to hit directed cycles whose length ℓ satisfies $(\ell \mod p = 1)$ for an integer $p \in \mathbb{N}$ given as input. Note that a fundamental difference between this result and Lemma 4.2.1 is that the latter works for any modulo and not just 1. The reason for this is explained in Section 4.6.

MODULO p DIRECTED CYCLE TRANSVERSAL (MOD(p)-DCT) **Parameter:** k **Input:** A digraph D and non-negative integers k and p. **Question:** Does there exist $S \subseteq E(D)$ of size at most k such that $D - S$ has no cycle of length 1 mod p?

Theorem 4.2.2. MOD(p)-DCT *on* $\mathcal{D}_\alpha$ *admits a kernel of size* $(p\alpha k)^{\mathcal{O}(4^{\alpha^3 p^2})}$.

4.3 Introducing Parity Preserving Sets

One of the main tools employed for the proof of Lemma 3.1.1 and Theorem 3.1.1 was Lemma 3.4.1, which says that if $\mathcal{Z}$ is a k-cut preserving set for (u, v) and A is a set of at most k arcs, then the existence of a (u, v)-path in $D - A$ implies the existence of one in $D[\mathcal{Z}] - A$. To prove Lemma 4.2.1 and Theorem 4.2.2, we need to take into account not only the existence of a path, but also its length modulo a certain integer p. For this reason, we introduce a notion of parity preserving sets. Vaguely speaking, in the a parity preserving set, like the cut preserving set, we need to keep witnesses for paths. In the case of cut preserving sets keeping a list of $k+1$ paths was enough to ensure the existence of one of them after the removal of at most k arcs, as one path from each of the lists, together would yield a walk which was sufficient for the cut-preserving purposes. In this case, since a walk doesn't necessarily yield the required parity path, we need to have witness lists of large enough size that ensure the existence of enough paths such that a disjoint witness of required parity can be found for each piece to yield a witnessing path for the given path. Also, the size of the list of paths required as a witness then also becomes a function of the size of the original path. This is formalized in the definition below.

Definition 4.3.1 ((k, p, q, t)-parity preserving set for a collection of pairs)**.** *For any digraph D, positive integers k, p, q, t, and a collection of pairs of vertices $\mathcal{V} \subseteq V(D)^2$, a set $\mathcal{Z}$ is called a* (k, p, q, t)-parity preserving set for $\mathcal{V}$ *if for any $A \subseteq E(D)$ of size at most k, if there exists $\mathcal{V}' \subseteq \mathcal{V}$, $|\mathcal{V}'| \leq q$, $\mathcal{V}' = \{(u_i, v_i) : i \in [q'], q' \leq q\}$ such that,*

1. *for each $(u_i, v_i) \in \mathcal{V}'$, there is a (u_i, v_i)-path P_i of length at most t in $D - A$,*
2. *for each $i, j \in [q']$, $i \neq j$, the paths P_i, P_j described above are internally vertex-disjoint,*
3. *for each $i \in [q']$, the internal vertices of P_i are disjoint from the set of vertices in the pairs of $\mathcal{V}$,*

then for each $i \in [q']$, there exists a (u_i, v_i)-path P_i^* in $D[\mathcal{Z}] - A$ such that,

1. *length of P_i modulo p is equal to length of P_i^* modulo p,*
2. *for each $i, j \in [q']$, $i \neq j$, the paths P_i and P_j^* are internally vertex-disjoint.*

4.3.1 Preserving Path Length Modulo p: The Ideas

The notion of cut preserving sets from the last chapter can also be used to preserve paths of certain length modulo some integer p under the failure of k arcs. The argument follows the same lines as the previous chapter; what we additionally need is the following observation. Suppose s and t are two vertices of a digraph $D \in \mathcal{D}_\alpha$ and there exists $p^2\alpha$ vertex-disjoint (s,t)-paths in D. By the pigeon hole principle, $p\alpha$ of those paths must have the same length modulo p. Let $P_1, \ldots, P_{p\alpha}$ denote those paths and X denote the set of vertices appearing just after s on these paths. The size of X is $p\alpha$, and since the largest independent set of D has size at most α, it means that the chromatic number of $D[X]$ is at least p. By Gallai-Roy Theorem [108, 189], there exists a path P of length $p-1$ in $D[X]$. Using this path and the path P_i, we can find a path of length i modulo p from s to t (see the proof of Lemma 4.4.1 for full details). This implies that if s and t are two vertices with more than $p^2\alpha + k$ vertex-disjoint paths from s to t, then preserving exactly $p^2\alpha + k$ of these paths is enough for our purpose. Indeed, after the removal of k arcs, there will still be $p^2\alpha$ vertex-disjoint (s,t)-paths and thus a path of every parity. This purpose can again be achieved using cut preserving sets.

4.4 Prelude to the Technical Details

Lemma 4.4.1. *Let $D \in \mathcal{D}_\alpha$ be a digraph and p be a positive integer. For $s, t \in V(D)$, if $\mathcal{P}$ is a collection of $p^2\alpha$ internally vertex-disjoint (s,t)-paths in D, then for each $i \in \{0, \ldots, p-1\}$, there exists an (s,t)-path of length $i \mod p$ in $D[V(\mathcal{P})]$.*

Proof. By the pigeonhole principle, there exist $p\alpha$ paths in $\mathcal{P}$ of the same length (without loss of generality, say 0) modulo p. Let $P_1, \ldots, P_{p\alpha}$ denote these paths. For each $j \in [p\alpha]$, let v_j be the vertex of P_j that appear after s in P_j. Since $\cup_{i \in [p\alpha]} P_i$ is a collection of internally vertex-disjoint paths, the set $X = \{v_1, \ldots, v_{p\alpha}\}$ is a set of $p\alpha$ vertices. Since $D[X] \in \mathcal{D}_\alpha$, it means that $\chi(D[X]) \geq p$ and thus, by Gallai-Roy Theorem [108, 189], there exists a path of length $p-1$ in $D[X]$. Without loss of

generality, let $P = (v_1, v_2, \ldots, v_p)$ be this path. Then for each $i \in [2..p]$, consider the path Q_i obtained as follows. Let R_1 be the (s, v_1)-subpath of P_1, that is R_1 is the arc (s, v_1). Let R_2 be (v_1, v_i)-subpath of P and let R_3 be the (v_i, t)-subpath of P_i. Then, $Q_i = R_1 \circ R_2 \circ R_3$ is an (s, t)-path. Clearly, the length of Q_i is $i - 1 \mod p$. □

The following proposition relates the absence of directed cycles in a strongly connected digraph with its absence in its undirected counterpart. This will be crucially used during the NP-hardness proof of DEOCT on tournaments.

Observation 4.4.1. *If D is a strongly connected digraph with no odd directed cycles, then the underlying undirected graph of D has no odd cycles, that is, it is bipartite.*

Proof. For the sake of contradiction, suppose that D is not bipartite, that is, it contains an undirected odd cycle, say C. Let $C = (v_0, \ldots, v_{t-1})$. If for all $i \in [t-1]$, $(v_i, v_{i+1}) \in E(D)$ (here addition is modulo t), then C is also a directed odd cycle in D, which is a contradiction. Otherwise, consider any pair v_i, v_{i+1}, such that $(v_{i+1}, v_i) \in E(D)$. Since D is a strongly connected digraph, there is path, say P_i from v_i to v_{i+1} in D. If P_i is an even path then P_i together with the arc (v_{i+1}, v_i) is an odd cycle in D (which is again a contradiction). Otherwise, for all pairs v_i, v_{i+1} such that $(v_{i+1}, v_i) \in E(D)$, there is an odd (v_i, v_{i+1})-path P_i. For each pair v_i, v_{i+1} such that $(v_{i+1}, v_i) \in E(D)$, replace the arc $(v_{i+1}, v_i) \in E(D)$ with the path P_i, in C, to obtain a directed closed odd walk. Since every directed closed odd walk contains a directed odd cycle, we conclude that D contains an odd cycle, which is a contradiction. □

4.5 Computing Parity Preserving Sets

We now show how to a find (k, p, q, t)-parity preserving sets using cut preserving sets. For this, first recall the definition of $\mathsf{dagify}(D, R)$ (Definition 3.5.1). Recall that, the strongly connected components of R are denoted by SC_i and their corresponding vertices in $R^\dagger$ are denoted by $SC_i^\dagger$.

Lemma 4.5.1. *Let k, p, q, t, ℓ be positive integers. Let $D \in \mathcal{D}_\alpha$ be a digraph, $S \subseteq V(D)$ and $R = V(D) - S$. Suppose that every strongly connected component of $D[R]$ contains at most ℓ vertices. Let $\beta = qt(2\ell^2\alpha + 1)p^2\alpha\ell^2 + (qt + 2k)\ell^2 + p^2\alpha$. Let $\mathcal{Z}^\dagger$ be a β-cut preserving set for S in $\mathsf{dagify}(D, R) = D^\dagger_R$, and let $\mathcal{Z} = S \cup \{SC_i \mid$*

there exists $v \in SC_i^\dagger$ such that $v \in \mathcal{Z}^\dagger$}. Then, $\mathcal{Z}$ is a (k,p,q,t)-parity preserving set for S^2 in D. Moreover, such a set $\mathcal{Z}$ of size at most $|S|^2 f(\beta, \alpha\ell^2)\ell^2$ can be computed, where f is as defined in Lemma 3.4.7.

Proof. From Observation 3.5.2, $D_R^\dagger \in \mathcal{D}_{\ell^2\alpha}$, which together with Lemma 3.4.7, implies that β-cut preserving set $\mathcal{Z}^\dagger$ for S in $D^{\dagger_R}$ of size at most $|S|^2 f(\beta, \alpha\ell^2)$ can be computed. Thus, $|\mathcal{Z}| \leq |S|^2 f(\beta, \alpha\ell^2)\ell^2$. It remains to show that $\mathcal{Z}$ is a (k,p,q,t)-parity preserving set for S^2 in D. For simplicity of notation, let $D^\dagger$ denote $D_R^\dagger$.

Recall the notation above Lemma 3.4.7. Notice that $\mathcal{Z}^\dagger$ is the union of β-cut preserving sets, specifically, a cut $C_{u,v}$ for (u^+, v^-) in $D^{\dagger}{}_{u,v}^{S}$ for all $u, v \in S$. For convenience, we will consider that any vertex $s \in S$ of $D_R^\dagger$ corresponds to the pair of vertices (s,s) of D. Because of this, any vertex v of $D_R^\dagger$ corresponds to a pair of vertices of D. In some arguments ahead, for a vertex (a_1, b_1) of $D_R^\dagger$, we will want to consider a (a_1, b_1)-path in the component of $D[R]$ containing a_1 and b_1. When $a_1 = b_1 = s \in S$, it will be always safe to take the single vertex s as this path.

Let $A \subseteq E(D)$ be of size at most k. Let $\{(u_1, v_1), \ldots, (u_c, v_c)\}$ be some pairs of vertices of S, where $c \leq q$ and let $P_1, \ldots, P_c$ be a set of paths such that,

1. for each $i \in [c]$, P_i is a (u_i, v_i)-path of length at most t in $D - A$,
2. for each $i, j \in [c]$, $i \neq j$, the paths P_i, P_j are internally vertex-disjoint,
3. for every $i \in [c]$, the internal vertices of P_i are disjoint from S.

We want to show the existence of a set of paths $P_1^*, \ldots, P_c^*$ such that every P_i^* is a (u_i, v_i)-path in $D[\mathcal{Z}] - A$, the length of P_i modulo p is the same as the length of P_i^* modulo p and, all the P_i^*'s are internally vertex-disjoint.

For each $i \in [c]$, let $P_i^\dagger$ be the reduced path of P_i in $D^\dagger$. Note that $P_i^\dagger$ is a (u_i, v_i)-path in $D^\dagger$. Since, $\mathcal{Z}^\dagger$ is a β-cut preserving set for (u_i, v_i) in $D^\dagger$, there exists a semi-$\mathcal{Z}^\dagger$-based partition of $P_i^\dagger$, $P_i^\dagger = P_{i,1}^\dagger \circ \cdots \circ P_{i,a_i}^\dagger$, such that for each $j \in [a_i]$, either $V(P_{i,j}^\dagger) \subseteq \mathcal{Z}^\dagger$ or there exists a list $\mathcal{L}_{i,j}^\dagger$ of size β containing paths between the same endpoints as $P_{i,j}^\dagger$.

Note 4.5.1. *For each $i \in [c]$, since the length of P_i is at most t, $a_i \leq t$.*

Claim 4.5.1. *Without loss of generality, for each $i \in [c], j \in [a_i]$, all the paths in $\mathcal{L}_{i,j}^\dagger$ have length at most $2\ell^2\alpha + 1$.*

Proof. Recall from Observation 3.5.2 that $D^\dagger \in \mathcal{D}_{\ell^2\alpha}$. Thus, from Observation 3.3.1, we can safely assume that all the paths in $\mathcal{L}^\dagger_{i,j}$ have length at most $2\ell^2\alpha + 1$. □

For each path $P^\dagger_{i,j}$ in $D^\dagger$, we will now associate a path $P_{i,j}$ in D with it. The path $P_{i,j}$ is basically the subpath of P_i with the endpoints corresponding to the endpoints of $P^\dagger_{i,j}$, that is, $P_{i,j}$ is obtained from $P^\dagger_{i,j}$ by replacing each vertex $\mathbf{v}^z_{x,y}$ in $P^\dagger_{i,j}$ by some (v_x, v_y)-path in SC_z. Consider the collection of paths $\{P_{i,j} \mid i \in [c], j \in [a_i]\}$. Observe that our goal reduces to finding another collection $\{P^*_{i,j} \mid i \in [c], j \in [a_i]\}$ such that,

1. the vertex sets of the internal vertices of $P^*_{i,j}$ and $P^*_{i',j'}$ are disjoint, for each $i, i' \in [c]$, $j \in [a_i]$, $j' \in [a_{i'}]$, $i \neq i'$, $j \neq j'$,
2. for each $i \in [c], j \in [a_i]$, the length of the path $P_{i,j}$ modulo p is the same as the length of the path $P^*_{i,j}$ modulo p.

It is enough to prove the above because the collection $P^*_i = P^*_{i,1} \circ \cdots \circ P^*_{i,a_i}$, for each $i \in [c]$ is the desired collection.

If $V(P^\dagger_{i,j}) \subseteq \mathcal{Z}^\dagger$, then let $P^*_{i,j} = P_{i,j}$. Note that, in this case, $V(P_{i,j}) = V(P^*_{i,j}) \subseteq \mathcal{Z}$. Thus, $P^*_{i,j}$ is a path in $D[\mathcal{Z}] - A$. In the other case, when there exists a list $\mathcal{L}^\dagger_{i,j}$ of paths of size β for $P^\dagger_{i,j}$, we clean this list as follows. First, since $c \leq q$, all the P_i's are of length at most t, each strongly connected component in $D[R]$ contains at most ℓ vertices and $|A| \leq k$, we need to remove only $(qt + 2k)\ell^2$ paths from each list $\mathcal{L}^\dagger_{i,j}$ so that the internal vertices of the remaining paths correspond to strongly connected components of $D[R]$ not containing any of the vertices in any of the P_i's or any vertex adjacent to any edge in A. Since the number of lists $\mathcal{L}^\dagger_{i,j}$ is bounded by ct (refer Note 4.5.1), and each path is smaller than $2\ell^2\alpha + 1$, we can remove at most $qt(2\ell^2\alpha + 1)p^2\alpha\ell^2$ further paths from each list so that the remaining lists, which we denote as $\widehat{\mathcal{L}}^\dagger_{i,j}$'s, consist of $p^2\alpha$ internally vertex-disjoint paths of $D\dagger^S_{u_i,v_i}$, such that if SC_r is a strongly connected component of $D[R]$ and a pair of vertices $(x_1, x_2) \in V(SC_r)$ is used in a path of $\widehat{\mathcal{L}^\dagger}_{i,j}$, then no other path of any other $\mathcal{L}_{x,q}$ can use a pair of vertices of $V(SC_r)$ as an internal vertex.

Let $Q^a_{i,j} = x_1 \circ x_2 \circ \cdots \circ x_f$ be a path of $\mathcal{L}_{i,j}$, with all the x_i's being vertices of C_{u_i,v_i}. We will construct a path $T^a_{i,j}$ of $D[\mathcal{Z}] - A$ as follows: x_1 is a vertex of $P^\dagger_i$ which corresponds in P_i to a path in some strongly connected component SC_r from some vertex a_1 to a vertex b_1. Let K_1 be this path. Likewise, x_f is a vertex of $P^\dagger_i$ which corresponds in P_i to a path in some strongly connected component $SC_{r'}$

from some vertex a_f to a vertex b_f. Let K_f be this path. Each of the other x_i's corresponds to a pair of vertices (a_i, b_i) in some strongly connected component SC_{r_i} of $D[R]$, let K_i be any (a_i, b_i)-path in this component. By definition of the digraph $D_R^\dagger$, it is clear that the concatenation of the K_i using the additional arcs from b_r to a_{r+1} for $r \in [f-1]$ is a (a_1, b_f)-path in $D[\mathcal{Z}]$. Moreover, because of the cleaning part of our argument, we know that none of the x_r for $r \in [2, f-1]$ belong to strongly connected component adjacent to arcs of S, which means that all the K_r for $r \in [2..f-1]$ are in $D[\mathcal{Z}] - A$. Since K_1 and K_f are subpaths of H_i, and don't contain any vertex not in C_{u_i,v_i}, they also belong to $D[\mathcal{Z}] - A$.

Let $Q_{i,j}^a$ be a path of some $\widehat{\mathcal{L}}_{i,j}^\dagger$, $Q_{i',j'}^{a'}$ a path of some $\mathcal{L}_{i',j'}$ (possibly $i = i'$ and $j = j'$) and $T_{i,j}^a$ and $T_{i',j'}^{a'}$ the paths of $D[\mathcal{Z}] - A$ associated. Because of the cleaning part of our procedure, the internal vertices of $Q_{i.j}^a$ and $Q_{i',j'}^{a'}$ belong to different strongly connected components of $D[R]$. This implies that the paths $T_{i,j}^a$ and $T_{i',j'}^{a'}$ are internally vertex-disjoint. For a similar reason, they are also internally vertex disjoint from all the vertices of all the other P_j. It means that, for any fixed i, j such that $P_{i,j}^\dagger$ does not belong to C_{u_i,v_i}, the only thing we have left to argue is that there exists, among the paths $T_{i,j}^a$, a path of the same length modulo p as $P_{i,j}^*$. This is done by Lemma 4.4.1, as there is at least $p^2\alpha$ of those paths. This ends the proof. □

4.6 Applications of Parity Preserving Sets

In this section we show how to utilize Lemma 4.5.1 to prove Lemma 4.2.1 and Theorem 4.2.2.

4.6.1 Fault-Tolerance (S, S)-Parity Reachability

In order to prove Lemma 4.2.1, we need a way to bound the size of the paths that we consider. This is the purpose of the next two lemmas.

Lemma 4.6.1. *Let $D \in \mathcal{D}_\alpha$, p be some positive integer and u and v be two vertices such that any strongly connected component in $D - \{u, v\}$ has size at most ℓ. If there exists a path P from u to v of length q such that $q \mod p = r$ for some $r \in [p-1]$ and $q \geq \alpha p\ell + 2$, then there exists a path P' from u to v of length q' such that $q' \mod p = (r+1)$ and $q - \alpha p\ell \leq q' < q$. Moreover, $V(P') \subset V(P)$.*

Proof. Suppose $P = x_0, \ldots, x_{q-1}$ and consider the set of vertices $S = \cup_{i \in [0,\alpha]} x_{1+ip\ell}$.

Because $D \in \mathcal{D}_\alpha$ and S is a set of size $\alpha + 1$, there is an arc between two vertices $x_{1+ip\ell}$ and $x_{1+jp\ell}$ for some $0 \leq i < j \leq \alpha$. This arc has to be oriented from $x_{1+ip\ell}$ to $x_{1+jp\ell}$ or it would create with the subpath of P from $x_{1+ip\ell}$ to $x_{1+jp\ell}$ a cycle of length greater than ℓ in $D - \{u, v\}$. Thus the path P' obtained from P by replacing the subpath of P from $x_{1+ip\ell}$ to $x_{1+jp\ell}$ by the arc $(x_{1+ip\ell}, x_{1+jp\ell})$ satisfies all the properties required. □

If P is a path of length at least $\alpha p^2 \ell$, it means we can apply Lemma 4.6.1 p times to get a path of the same parity. This gives the following lemma.

Lemma 4.6.2. *Let D be a digraph in $\mathcal{D}_\alpha$, p some positive integer and u and v two vertices such that any strongly connected component in $D - \{u, v\}$ has size at most ℓ. If there exists a path P from u to v of length q such that $q \mod p = r$ for some $r \in [p-1]$, then there exists a path P' from u to v of length q' such that $q' \mod p = r$ and $q' \leq \alpha p^2 \ell + 2$. Moreover, $V(P') \subset V(P)$.*

We are now ready to prove the parity version of our fault-tolerant result.

Lemma 4.2.1. *Given a digraph $D \in \mathcal{D}_\alpha$, positive integers k, ℓ, p, a non-negative integer r, and $S \subseteq V(D)$ such that every strongly connected component of $D - S$ has at most ℓ vertices, the* Fault-Tolerance (S, S)-Parity Reachability *problem admits a solution H on $(|S|\alpha\ell pk)^{\mathcal{O}(4^{\alpha\ell^2})}$ vertices. Furthermore, such a solution H can be found in polynomial time.*

Proof. Let $\mathcal{Z}'$ be a $(k, p, |S|, \alpha p^2 \ell + 2)$-parity-preserving set obtained by applying Lemma 4.5.1 to D and S. A rough computation would show that the β defined in Lemma 4.5.1 is then smaller than $20|S|\alpha^3\ell^5 p^5 + 2k\ell^2$, which gives that:

$$|\mathcal{Z}'| \leq |S|^2 \ell^2 (22(20|S|\alpha^3\ell^5 p^5 + 2k\ell^2)^5)^{4^{\alpha\ell^2}}.$$

Let us now show that $\mathcal{Z}'$ is a solution to the Fault-Tolerance (S, S)-Parity Reachability problem. Let A be a set at most k arcs, s and t two vertices of S, and P a path from s to t in $D - A$. Let $s_1, \ldots, s_e$ denote the vertices in the intersection of P with S, in the order as they appear on P, and for every $i \in [e-1]$, P_i denote the subpath of P from s_i to s_{i+1}. As P is a path, $e \leq |S|$. By applying Lemma 4.6.2, we can assume that all the P_i's are smaller than $\alpha p^2 \ell + 2$. Thus, by definition of a $(k, p, |S|, \alpha p^2 \ell + 2)$-parity-preserving set, for every $i \in [e]$ there exists a path P_i' in $D[\mathcal{Z}'] - A$ from s_i to s_{i+1} of the same length modulo p as P_i and such

that all the P'_i are internally vertex-disjoint. Taking the union of the P'_i gives the desired (s,t)-path in $D[\mathcal{Z}'] - A$. □

4.6.2 Kernel for MOD(p)-DCT in $\mathcal{D}_\alpha$

In this section, we present a proof of Theorem 4.2.2. The proof follows the same structure as the proof of Theorem 3.1.1. First we need to find an approximate solution of polynomial size. For this we need the following result, due to Chen et al. [52]

Theorem 4.6.1. *Let $l \geq 2$ be an integer. If a strongly connected digraph D contains no directed cycle of length $1 \mod p$, then $\chi(D) \leq p$.*

Remember that, when $D \in \mathcal{D}_\alpha$, $\chi(D) \leq p$ implies that $|D| \leq \alpha p$.

Lemma 4.6.3. *Let $D \in \mathcal{D}_\alpha$ and p a positive integer. Then either D does not contain a cycle of length $1 \mod p$, or such a cycle on fewer than $p(\alpha+1)^2$ vertices exists.*

Proof. Suppose C is the smallest cycle of length $1 \mod p$, and $|C| = q \geq (\alpha+1)^2 p$. Let $C = x_0, \ldots, x_{q-1}$ denote the vertices of C. Consider the set of vertices $A = \cup_{i \in [0,\alpha]} \{x_{i\alpha p}\}$. Because A contains more than α vertices, there is an arc between two vertices of A, say from $x_{i\alpha p}$ to $x_{j\alpha p}$. However, since $q \geq (\alpha+1)^2 p$, the subpath of C from $x_{j\alpha p}$ to $x_{i\alpha p}$ contains more than αp vertices. Let C' denote the vertices on this path. $D[C']$ is a strongly connected graph on more than αp vertices. Theorem 4.6.1 implies the existence of a cycle of length $1 \mod p$ in $D[C']$ which is a contradiction as C' is smaller than C. □

With the previous lemma, one can easily adapt the proof of Lemma 3.5.1 to show the following:

Lemma 4.6.4. *Let (D,k) be an instance of* MOD(p)-DCT *and let $D \in \mathcal{D}_\alpha$. In polynomial time, one can either correctly conclude that (D,k) is a* NO *instance of* MOD(p)-DCT*, or output a set $S \subseteq V(D)$ such that $|S| \leq (\alpha+1)^2 pk$ and $D - S$ does not have any cycle of length $1 \mod p$.*

We are now ready to prove the existence of a kernel for MOD(p)-DCT:

Theorem 4.2.2. MOD(p)-DCT *on $\mathcal{D}_\alpha$ admits a kernel of size $(p\alpha k)^{\mathcal{O}(4^{\alpha^3 p^2})}$.*

Proof. Let (D, k) be an instance MOD(p)-DCT. By applying Lemma 4.6.4, we can either find $k+1$ vertex disjoint cycles of length $1 \mod p$, and conclude that (D, k) is a NO instance, or find a set S of size at most $k(\alpha+1)^2 p$ vertices such that $D - S$ doesn't contain any cycle of length $1 \mod p$. Let $R = V(D) - S$ and note that Theorem 4.6.1 implies that the strongly connected component of $D[R]$ have at most αp vertices. Let $\mathcal{Z}$ be a $(k, p, p(\alpha+1+k)^2, p(\alpha+1+k)^2)$-parity preserving set for S obtained from applying Lemma 4.5.1 to D and S. Note that, the β defined in Lemma 4.5.1 is then smaller than $10p^8\alpha^{10}k^4$ and thus $|\mathcal{Z}| \leq (k(\alpha+1)^2 p)^2(22(10p^8\alpha^{10}6k^4)^5)^{4^{\alpha^3 p^2}}$. We claim that (D, k) is a YES instance of MOD(p)-DCT if and only if $(D[\mathcal{Z}], k)$ is a YES instance of MOD(p)-DCT.

For the ease of notation, let $D' = D[\mathcal{Z}]$. Since D' is a subgraph of D, the forward direction is trivial. For the backward direction, let A be a set of at most k arcs such that of $D' - A$ has no cycle of length $1 \mod p$. We will now prove that $D' - A$ also has no cycle of length $1 \mod p$. For the sake of contradiction, suppose there is a cycle of length $1 \mod p$ in $D - A$ and let C be the smallest such cycle. Since $D \in \mathcal{D}_\alpha$ and $|A| \leq k$, $D - A \in \mathcal{D}_{\alpha+k}$. Then by Lemma 4.6.3, the length of C is at most $p(\alpha+1+k)^2$.

Since $D[R]$ has no cycle of length $1 \mod p$, $C \cap S \neq \emptyset$. Let $v_1, \ldots, v_q$ be the vertices of $C \cap S$ in the order as they appear on C. Note that $q \leq p(\alpha+1+k)^2$. Then, for each $i \in [q]$, there is a subpath $P_{i,i+1}$ (count $q+1$ as 1) from v_i to v_{i+1}, such that $P_{i,i+1}$ is S-free. Note that all the length of each of these paths is at most the length of the cycle C which is at most $p(\alpha+1+k)^2$.

Claim 4.6.1. *For each $i \in [q]$, there exists a path $P'_{i,i+1}$ from v_i to v_{i+1} in $D' - A$ such that:*

1. *the length modulo p of $P'_{i,i+1}$ is the same as the one of $P_{i,i+1}$, and*
2. *for any $i, j \in [q]$, $i \neq j$, the set of internal vertices of $P'_{i,i+1}$ and $P'_{j,j+1}$ are disjoint.*

Proof. Since $\mathcal{Z}$ is a $(k, p, p(\alpha+1+k)^2, p(\alpha+1+k)^2)$-parity-preserving set for S^2 in D and $\bigcup_{i\in[q]}(v_i, v_{i+1}) \subseteq S^2$, the claim follows from the definition of $(k, p, p(\alpha+1+k)^2, p(\alpha+1+k)^2)$-parity-preserving set for S^2. □

Consider the cycle C' formed by taking the union of all the paths $P'_{i,i+1}$, for all $i \in [q]$. From Claim 4.6.1, C' exists in $D' - A$ and has the same length modulo p as C. This contradicts the definition of A and ends the proof. □

Chapter 5

Polynomial Kernel for DFVS Parameterized by the Solution Size Plus the Size of a Treewidth-η Modulator

Recently, in order to shed some light on the kernelization complexity of DFVS, Bergougnoux et al. [19] studied the kernelization complexity of DFVS where, in addition to the solution size, they parameterized the problem by the size of the feedback vertex set of the underlying undirected graph (that is, a modulator to a graph of treewidth 1). In this chapter, we give a polynomial kernel for DFVS parameterized by the solution size plus the size of a treewidth-η modulator, for any fixed integer η. Formally, for a directed graph D, a subset $M \subseteq V(D)$ is called a *treewidth η-modulator* if $D - M$ has treewidth at most η. We consider the following parameterized problem parameterized by $k + \ell$.

DFVS/DFVS+Treewidth-η Modulator (Tw-η Mod)
Input: A digraph D, $k \in \mathbb{N}$, $M \subseteq V(D)$ where $|M| = \ell$ and $D - M$ has treewidth at most η.
Question: Is there $S \subseteq V(D)$ where $|S| \leq k$ and $D - S$ is a DAG?

Observe that DFVS/DFVS+Tw-η Mod is the same problem as DFVS with just a different parameter. We design a kernel for DFVS/DFVS+Tw-η Mod parameterized by $k + \ell$. An additional feature of our kernelization algorithm is that given an instance (D, M, k) of DFVS/DFVS+Tw-η Mod, in polynomial time, it

outputs an equivalent instance $(D', M., k')$ of DFVS/DFVS+Tw-η Mod such that the size of D' is a polynomial function in k and ℓ (assuming fixed η), $M' \subseteq M$ and $k' \leq k$. In literature, such kernels, where the parameter of the reduced instance does not increase (implied by assuming that $M' \subseteq M$ and $k' \leq k$) are called *proper kernels* [2]. The additional requirement of the non-increase in the parameter of the reduced instance is backed by the intuition that kernelization algorithms are meant to output a smaller instance (compared to the input instance), and an increase in even the parameter would counter this intuition. Thus, proper kernels are a neat formalization of the heuristic based idea of pre-processing that has been extremely successful in practice. Our main contribution is the following theorem.

Theorem 5.0.1. DFVS/DFVS+Tw-η Mod *admits a proper kernel of size* $(k \cdot \ell)^{\mathcal{O}(\eta^2)}$.

Notably, our result can be viewed as a proof that DFVS parameterized *only* by k, admits a polynomial kernel on the class of *all* graphs whose treewidth can be made constant by the removal of $k^{\mathcal{O}(1)}$ vertices. Yet another justification for our choice of the parameter is the following. Parameterized by k alone, the problem has been open for a very long time. On the other hand, parameterized by ℓ alone, it can be easily seen that unless NP $\subseteq$ coNP/poly, the problem does not exhibit a polynomial kernel. This follows from an easy reduction from Vertex Cover parameterized by the size of a treewidth-2 modulator, which is shown to not admit a polynomial kernel unless NP $\subseteq$ coNP/poly [68]. Thus, $k + \ell$ is a natural parameter to explore.

We also remark that the proof of Theorem 5.0.1 required the development of a novel use of important separators, among other ideas for finding protrusions and using the state-of-the-art protrusion machinery. Thus, as a side reward, the ideas developed in this article may be insightful, helping to design reduction rules for a polynomial kernel of DFVS. Lastly, our result encompasses the recent result of Bergougnoux et al. [19], where they studied DFVS parameterized by the feedback vertex set number of the underlying undirected graph, and gave a polynomial kernel for this problem. Specifically, they gave a kernel of size $\mathcal{O}(\mathbf{fvs}^4)$, where **fvs** is the feedback vertex set number of the underlying undirected graph of D. Note that our parameter $k + \ell$ is not only upper bounded by $\mathcal{O}(\mathbf{fvs})$, but it can be arbitrarily smaller than **fvs**. We also remark that DFVS has already been parameterized by treewidth in the literature (not for kernelization purposes)—recently, Bonamy et al. [33] showed that DFVS parameterized by the treewidth of the input graph, t, can be solved in time $2^{\mathcal{O}(t \log t)} n^{\mathcal{O}(1)}$, and that unless the Exponential Time Hypothesis fails, it cannot be solved in time $2^{o(t \log t)} n^{\mathcal{O}(1)}$.

5.1 Overview of the Kernelization Algorithm

Our kernelization algorithm of Theorem 5.0.1 can be divided into three main phases. We give a brief summary of each phase here.

1. Computing a zone decomposition of the directed graph: We first compute a decomposition of D into three components: the vertex set M (modulator), a collection of $\mathcal{O}(k\ell^2)$ vertex sets $\mathcal{Z}$ (zones), and a vertex set R (remainder) of size $\mathcal{O}(k\ell^2)$. All of these sets are pairwise disjoint and form a partition of $V(D)$. The aim of this decomposition is to achieve a few properties with respect to each zone $Z \in \mathcal{Z}$, which we will later exploit to design reduction rules to bound the size of each zone. Since the number of zones in the decomposition and the size of R is $\mathcal{O}(k\ell^2)$, in order to get the desired kernel, it would be enough to bound the size of each zone $Z \in \mathcal{Z}$ by $k\ell^{\mathcal{O}(\eta^2)}$, after such a decomposition is constructed.

Let us mention three important properties of a zone $Z \in \mathcal{Z}$ that this decomposition achieves, and which play a critical role in helping us bound the size of Z. The first property is that if D has a directed feedback vertex set of size at most k, then there exists a directed feedback vertex set, say S, in D of size at most k, whose intersection with Z is of constant size. The second property is that the neighborhood of Z is entirely contained in $M \cup R$ and the size of the neighborhood of Z in R is bounded by some constant. Finally, for any two vertices in the neighborhood of Z in M, the maximum value of a directed flow from one to the other is either extremely high or zero. We will exploit the first property to mark a "small" set of vertices in Z that in some sense "represents" all partial solutions in Z. Such a set, which is called Γ_{DFVS}, is then used to design reduction rules that eliminate arcs between $Z \setminus \Gamma_{\text{DFVS}}$ and M. The third property is critically used in these reduction rules. Then, from the second property, all the vertices in $Z \setminus \Gamma_{\text{DFVS}}$ have a constant-sized neighborhood outside Z. Having this information at hand, we further partition Z into small slices, each of which is then replaced by constant sized sets by using protrusion machinery.

Though at first glance this decomposition of the graph may look very similar to the near-protrusion decomposition of [99], it is not a near-protrusion decomposition. In fact, for this problem we cannot find a near-protrusion decomposition.

2. Computing a k-DFVS Representative Set in Z: A k-DFVS representative in a zone Z is a subset of vertices of Z (say, Γ_{DFVS}) with the following property: If D has a directed feedback vertex set of size at most k, then there is a directed feedback vertex set S in D of size at most k and $S \cap Z \subseteq \Gamma_{\text{DFVS}}$. We aim to compute such

a set whose size is bounded by some polynomial in k and ℓ. For this purpose, we first revisit the relation between our problem and Skew Multicut. In particular, we see that for any directed feedback vertex set S in D, $S \cap Z$ is a solution to an "appropriate" instance of the Skew Multicut problem. Thus, if we can compute solutions to all possible appropriate instances of Skew Multicut, then we can set Γ_{DFVS} to be the union of all these solutions. In this overview, we prefer to keep the notion of an appropriate instance abstract.

Unfortunately, a single instance of our problem gives rise to a huge number of appropriate instances. In particular, if we naively construct Γ_{DFVS} by *individually* computing a solution for each possible choice for an appropriate instance, we do not obtain a set whose size is bounded by a polynomial function in k and ℓ. So, in the second step, we invest significant efforts to construct a set Γ_{DFVS} of the desired small size, which contains a solution for each possible choice of an appropriate instance. To this end, we observe that if such a set of the desired size exists, then solutions of "many" possible appropriate instances intersect a lot. Very roughly speaking, we aim to identify a small set of vertices that is guaranteed to be contained in solutions of "many" instances. If we can identify such a set, then we delete it from all appropriate instances in which it is guaranteed to be present in some solution, and recurse on the resulting instances. (Here, only one recursive call is performed.) From the properties of a zone decomposition, we are able to derive that there is a solution to our original problem whose intersection with Z is small, which in turn leads us to the observation that we can only focus on small solutions for each appropriate instance. Hence, we can bound the depth of the recursion. Though this description roughly conveys the broad picture, the implementation of these ideas is significantly more complex. For example, we are unable to find a small set of vertices that is contained in some solution for "many" instances. Instead, we find a *collection* of small sets such that at least one among them is the set that we want, though we do not know which one.

These abstract ideas are materialized with the help of important separators (defined in Section 5.2), the Pushing Lemma for Skew Multicut and a new (simple) lemma, which we call the *Important Separator Preservation (via Small Sink Set) Lemma*. This lemma says that if S is an important (X, Y)-separator of size α in some digraph, then S is also an important (X, Y')-separator for some subset Y' of Y of size at most $\alpha + 1$, where $Y \setminus Y'$ is removed from that digraph. We mainly use this lemma in situations (that arise when we try to compute the collection of sets mentioned above) that require guessing the set Y when X is given, so that we

can compute important (X, Y)-separators. In these situations, since it is enough to guess Y' to compute all important (X, Y)-separators (from the Important Separator Preservation Lemma), the fact that Y' is small significantly reduces the search space for Y' compared to that for Y.

3. Reduction Rules for bounding the size of each zone Z: After computing a small set that is a k-DFVS representative in a zone in our zone decomposition, our final objective is to bound its size. To this end, we design reduction rules that decompose a zone Z into a "small" number of *protrusions*. (Roughly speaking, a protrusion in a graph G is an induced subgraph $G[U]$ of G for a subset $U \subseteq V(G)$ that has constant treewidth and only a constant number of vertices with neighbors in $G - U$.) More precisely, we first design a set of reduction rules that only bounds the size of the neighborhood of every zone Z (with Γ_{DFVS} removed) outside Z by a constant. Then, by computing a nice tree decomposition of $D[Z]$ and relying on properties of an LCA-closure in that tree, we decompose the set Z as $Z = \widetilde{\Gamma}_{\text{DFVS}} \uplus \biguplus_{U \in \mathcal{U}} U$ such that $\Gamma_{\text{DFVS}} \subseteq \widetilde{\Gamma}_{\text{DFVS}}$, the size of $\mathcal{U}$ is "small", and each set $U \in \mathcal{U}$ induces a protrusion. We then replace each protrusion $D[U]$ by a "small" digraph such that the resulting digraph is a "minor" of the original digraph, and the input modulator M is also a treewidth-η modulator in the resulting digraph. This concludes our kernelization algorithm.

We remark that the operations of the last step of our kernelization algorithm ensure that the input modulator M remains a modulator in the final returned kernel. If we allow the modulator in the returned instance to be of size larger than $|M|$ (that is, if we allowed non-proper kernel), then we can bypass all of these reduction rules and the protrusion machinery, and directly create the kernelized instance by taking the torso of the set S that is the union of M, R and a k-DFVS representative set in each Z. Here, by torso we mean that for every two vertices $u, v \in S$ with a directed path from u to v whose internal vertices do not belong to S, we add an arc from u to v. Since the set S is of small size (polynomial in k and ℓ), we directly obtain a kernel by omitting the vertices outside S. However, when we perform the torso operation, we lose the property that M is a modulator for the final instance, which means that the parameter can increase to be of the magnitude of the entire kernel.

5.2 Prelude to the Technical Details

For any $X, Y \subseteq V(D)$, an (X,Y)-*separator* in D is a set of vertices, say $S \subseteq V(D) \setminus (X \cup Y)$, such that there is no path from any vertex in X to any vertex in Y in $D - S$. Notice that the separator should have an empty intersection with $X \cup Y$.

Let D be a directed graph and let $V(D) = \{v_1, \ldots, v_n\}$. A *topological ordering* of D is an ordering π of $V(D)$ such that if $(v_{\pi(i)}, v_{\pi(j)}) \in E(D)$, then $i > j$. **We stress that here we suppose that no arc is directed from a vertex v to a vertex u that occurs before v, while it might be more standard to consider the symmetric condition.** It is well known that a digraph has a topological ordering if and only if it is acyclic. A set $S \subseteq V(D)$ whose deletion makes the digraph acyclic is called a *directed feedback vertex set* of D. Given a topological ordering π of D, for any $X \subseteq V(D)$, we say π_X is an ordering induced by π if the vertices of X appear in the same order in π_X and in π.

LCA-closure. For a rooted tree T and a subset $M \subseteq V(T)$, the least common ancestor-closure of M, **LCA-closure**(M) is obtained by the following process. Initially, set $M' = M$. As long as there are vertices x and y in M' whose least common ancestor w is not in M', add w to M'. When the process terminates, output M' as the LCA-closure of M.

Proposition 5.2.1 (Lemma 1, [99])**.** *Let T be a rooted tree, $M \subseteq V(T)$ and $M' =$* **LCA-closure**(M)*. Then $|M'| \leq 2|M|$ and for every connected component C of $T - M'$, $|N(C)| \leq 2$.*

5.3 Decomposing the Graph

Let (D, k, M) be an instance of DFVS/DFVS+Tw-η Mod. Informally, the objective of this section is to compute a decomposition of D that consists of three components: the vertex set M (modulator), a collection of vertex sets $\mathcal{Z}$ (zones), and a vertex set R (remainder). All of these sets would be pairwise disjoint. The crux is to "divide-and-conquer" D so that each zone—that is, a set $Z \in \mathcal{Z}$—would correspond to a subproblem that is easier to solve than (D, k, M) because (1) the intersection of a minimum solution with Z would be necessarily small, and (2) the interaction of Z is "well-structured" with respect to M, "limited" with respect to R , and "non-existent" with respect to any other zone.

Towards the computation of R, we need to compute three sets: (i) a solution S in $D-M$; (ii) a set F to separate vertices in M that have low-flow; (iii) an LCA-closure of bags derived from $S \cup F$. The arguments we provide along the way to construct these sets will only partially prove that we have indeed derived the decomposition we desire. At the end, we complete the proof by focusing on the property regarding the intersection of a minimum solution with each zone. In what follows, we execute this plan.

Phase I: Solution S. The first phase of our proof is simple. Recall that the treewidth of $D - M$ is upper bounded by η. Furthermore, Bonamy et al. [33] have shown that, given a (di)graph D' of treewidth t, a smallest size set S' such that $D' - S'$ is a DAG can be computed in time $2^{\mathcal{O}(t \log t)} n^{\mathcal{O}(1)}$. In particular, this means that in time $2^{\mathcal{O}(\eta \log \eta)} n^{\mathcal{O}(1)}$—that is, polynomial time—we can compute a subset $S \subseteq V(D)$ of smallest size such that $D - (M \cup S)$ is a DAG. In case $|S| > k$, we directly conclude that (D, k, M) is a No instance of DFVS/DFVS+Tw-η Mod. Otherwise, we proceed as described below.

Phase II: Flow-Blocker F. The purpose of the second phase is to compute a subset $F \subseteq V(D)$ that governs the flow between *all* pairs of vertices in M. Having this subset at hand will be crucial when we later argue about the intersection of a minimum directed feedback vertex set with each zone. Specifically, for every pair of vertices in M, if we can easily separate them—we simply do that; otherwise, no solution of size at most k can separate these vertices, which is again beneficial for our arguments. Let us now proceed to the formal description of the computation of F.

Formally, for every ordered pair (u, v) of (not necessarily distinct) vertices $u, v \in M$ such that $(u, v) \notin E(D)$, we compute a subset $C_{(u,v)} \subseteq V(D) \setminus ((M \cup S) \setminus \{u, v\})$ of minimum size such that $D - ((M \cup S \cup C_{(u,v)}) \setminus \{u, v\})$ has no directed path from u to v (that consists of more than one vertex), that is, $C_{(u,v)}$ is a vertex cut. The computation of each such set $C_{(u,v)}$ can be executed in polynomial time by using, for example, Ford-Fulkerson algorithm [104]. Having all vertex cuts at hand, we define $F = \bigcup_{(u,v) \in M \times M \setminus E(D), |C_{(u,v)}| \leq k} C_{(u,v)}$. In words, we take the union of all cuts $C_{(u,v)}$ of size at most k. Note that $|F| \leq k\ell^2$.

Before we turn to further enriching the set $S \cup F$, let us formally summarize the structural properties already induced by this set on $D - (M \cup S \cup F)$. To this end, note that by the classic Menger's theorem, the size of $C_{(u,v)}$ for a pair $(u, v) \in M \times M \setminus E(D)$ equals the number of internally vertex-disjoint paths from

u to v in $C_{(u,v)} \subseteq V(D) \setminus (M \cup S \cup \{u, v\})$. Thus, the correctness of our observation is an immediate consequence of the construction of S and F.

Observation 5.3.1. *The digraph $D - (M \cup S \cup F)$ is a DAG such that for every pair $(u, v) \in M \times M \setminus E(D)$, either there is no path from u to v in the digraph $D - ((M \cup S \cup F) \setminus \{u, v\})$, or there are at least $k+1$ internally vertex-disjoint paths from u to v in D.*

Phase III: LCA-Closure to Derive R. Having that the set $M \cup S \cup F$ is not sufficient for us—the vertices of $D - (M \cup S \cup F)$ can have many neighbors in $M \cup S \cup F$ while Observation 5.3.1 provides us a handle only for those neighbors in M. However, we can compute a subset $R \subseteq V(D) \setminus M$ that contains $S \cup F$, such that no vertex of $D - (M \cup R)$ has many neighbors in R. In fact, we require a stronger claim—we will bound the neighborhood not only of each individual vertex, but also of entire sets of vertices that will later be called zones. For this purpose, we rely on the fact that the treewidth of $D - M$ is at most η. Specifically, we will "highlight" a small set of bags (in a tree decomposition) that captures $S \cup F$, and since bags are separators and the size of each bag is small, we will be able to derive the desired set R.

Towards the computation of R, we first obtain a nice tree decomposition (T, β) of $D - M$ of width at most η. This step is done in time $\mathcal{O}(n)$ using the algorithms of Propositions 2.5.2 and 2.5.3 because $\eta = \mathcal{O}(1)$. Now, we highlight bags that capture the vertices in $S \cup F$ as follows. For every vertex $v \in S \cup F$, select (arbitrarily) a node $t_v \in V(T)$ such that $v \in \beta(t)$. Then, we define B as the set of all selected nodes, that is, $B = \{t_v : v \in S \cup F\}$. Next, keeping in mind that our eventual goal is to control sizes of neighborhoods, we define $B^\star$ as the LCA-closure of B in T. From Proposition 5.2.1, $|B^\star| \leq 2 \cdot |B| \leq 2(|S| + |F|) \leq 2k(\ell^2 + 1)$. Having $B^\star$ at hand, we define R as the union of all bags corresponding to its vertices, that is, $R = \bigcup_{t \in B^\star} \beta(t)$. Then, the following observation is immediate.

Observation 5.3.2. $|R| \leq 2k(\eta + 1)(\ell^2 + 1)$.

Decomposition. Having already computed R, it remains to partition $V(D) \setminus (M \cup R)$ into zones. To this end, we first define $\mathcal{C}$ as the set of connected components (subtrees) of the forest $T - B^\star$. Since (T, β) is a nice tree decomposition, the degree of every node is at most 3, which means that $|\mathcal{C}| \leq 3|B^\star| \leq 6k(\ell^2 + 1)$. Now, for every tree $C \in \mathcal{C}$, we define $Z_C = \bigcup_{t \in V(C)} \beta(t)$ and $Z_C^\star = Z_C \setminus R$. Finally, the collection of zones is given by $\mathcal{Z} = \{Z_C^\star : C \in \mathcal{C}\}$. Because (T, β) is a tree

decomposition, for every vertex $v \in V(D)$, the set of nodes whose bags contain v induce a tree, and hence it is immediate that for all distinct $C, C' \in \mathcal{C}$, we have that $Z^\star_C \cap Z^\star_{C'} = \emptyset$. Furthermore, because (T, β) is a tree decomposition, for every arc $(u, v) \in E(D - M)$, there exists $t \in V(T)$ such that $u, v \in \beta(t)$, and hence it is also immediate that for all $Z^\star_C \in \mathcal{Z}$, $N(Z_C) \cap R$ is a subset of the bags of the nodes in $N(C)$ (that is, the neighbors of the nodes in C in T); because we have defined $B^\star$ as the LCA-closure of B, $|N(C)| \leq 2$ (see Proposition 5.2.1). Specifically, this means that for all $Z^\star_C \in \mathcal{Z}$, $N(Z^\star_C) \subseteq M \cup R$ and $|N(Z^\star_C) \cap R| \leq 2(\eta + 1)$.

Let us summarize the properties of our decomposition, as well as its construction, in the following definition and lemma.

Definition 5.3.1. *Let (D, k, M) be an instance of* DFVS/DFVS+Tw-η Mod. *A partition $V(D) = M \uplus R \uplus (\biguplus_{Z \in \mathcal{Z}} Z)$ is a* zone-decomposition *if:*

1. *$D - (M \cup R)$ is a DAG.*
2. *For all $Z \in \mathcal{Z}$, we have $N(Z) \subseteq M \cup R$, and $|N(Z) \cap R| \leq 2(\eta + 1)$.*
3. *For all $(u, v) \in M \times M \setminus E(D)$, the digraph $D - ((M \cup R) \setminus \{u, v\})$ either has at least $k + 1$ internally vertex-disjoint paths from u to v, or it has no path from u to v (that consists of more than one vertex).*

Lemma 5.3.1. *There is a polynomial-time algorithm that, given an instance (D, k, M) of* DFVS/DFVS+Tw-η Mod, *either correctly decides that (D, k, M) is a* No*instance, or constructs a zone-decomposition $V(D) = M \uplus R \uplus (\biguplus_{Z \in \mathcal{Z}} Z)$ with $|\mathcal{Z}| \leq 6k(\ell^2 + 1)$ and $|R| \leq 2(\eta + 1)k(\ell^2 + 1)$.*

Small Intersection Property. We are now ready to argue that if (D, k, M) is a Yes instance, then the size of the intersection of every minimum(-size) solution with every zone is merely a constant. Formally, we have the following lemma.

Lemma 5.3.2. *Let (D, k, M) be a* Yes *instance of* DFVS/DFVS+Tw-η Mod *with zone-decomposition $V(D) = M \uplus R \uplus (\biguplus_{Z \in \mathcal{Z}} Z)$. For any minimum-sized directed feedback vertex set S of D, we have $|S \cap Z| \leq |N(Z) \cap R| \leq 2(\eta + 1)$ for all $Z \in \mathcal{Z}$.*

Proof. Let S be a directed feedback set of D of minimum size. Since (D, k, M) is a Yes instance, we have that $|S| \leq k$. Suppose, by way of contradiction, that there exists $Z \in \mathcal{Z}$ such that $|S \cap Z| > |N(Z) \cap R|$. Let $S^\star = (S \setminus Z) \cup (N(Z) \cap R)$. Clearly, $|S^\star| < |S|$. However, this means that $S^\star$ is not a directed feedback vertex set of D.

Thus, there exists a directed cycle C in $D - S^\star$. Note that $V(C) \cap (N(Z) \cap R) = \emptyset$ because $N(Z) \cap R \subseteq S^\star$.

Since $D - S$ is a DAG and $S \setminus Z \subseteq S^\star$, we have that $V(C) \cap Z \neq \emptyset$. Moreover, $D[Z]$ is a DAG since $D - (M \cup R)$ is a DAG (by Definition 5.3.1). Because $V(C) \cap (N(Z) \cap R) = \emptyset$ and $N(Z) \subseteq M \cup R$ (by Definition 5.3.1), this means that $V(C) \cap (N(Z) \cap M) \neq \emptyset$. In turn, this means that the directed cycle C is of the form $C = u_1 - P_{u_1,u_2} - u_2 - P_{u_2,u_3} - \ldots - u_{r-1} - P_{u_{r-1}u_r} - u_r$ for some $r \geq 1$, such that $u_1 = u_r$, $u_1, \ldots, u_r \in N(Z) \cap M$, and for all $i \in \{1, \ldots, r\}$, $u_i - P_{u_iu_{i+1}} - u_{i+1}$ is a directed path from u_i to u_{i+1} in D whose internal vertices either all belong to Z or all belong to $V(D) \setminus (M \cup R \cup Z)$. Consider any non-empty path $P_{u_iu_{i+1}}$ where all the vertices belong to Z. By Definition 5.3.1 (Condition 3), there are at least $k+1$ internally vertex-disjoint paths from u_i to u_{i+1}. Since $|S| \leq k$, there exists a path, say $u_i - P'_{u_iu_{i+1}} - u_{i+1}$, all of whose internal vertices do *not* belong to S. This means that we can replace (in C) each non-empty path $P_{u_iu_{i+1}}$ whose vertices belong to Z by the corresponding path $P'_{u_iu_{i+1}}$ as defined above, thereby getting a directed closed walk C' in $D - S$. As a directed closed walk contains a directed cycle, this contradicts the choice of S as a directed feedback set of D. □

Important Note. From now onwards, we denote by $V(D) = M \uplus R \uplus (\biguplus_{Z \in \mathcal{Z}} Z)$ a zone-decomposition as computed by the algorithm of Lemma 5.3.1. Recall that $|M| = \ell$, $|\mathcal{Z}| \leq 6k(\ell^2+1)$ and $|R| \leq 2(\eta+1)k(\ell^2+1)$, and thus to derive our polynomial kernel, it is next sufficient to upper bound the size of each set $Z \in \mathcal{Z}$ individually. For this purpose, from now onwards, we fix an arbitrary set $Z \in \mathcal{Z}$, and argue how the size of Z can be bounded. As the choice of Z is arbitrary, we can thus bound the size of every set in $\mathcal{Z}$. When we eventually formally prove Theorem 5.0.1, we shall zoom out of the view of a specific $Z \in \mathcal{Z}$, but until we reach this (short) proof, we suppose that Z is fixed.

5.4 Reducing Each Part: k-DFVS Representative Marking

For an instance (D, k, M) of DFVS/DFVS+Tw-η Mod, the kernelization algorithm starts by applying Lemma 5.3.1 and either concludes that (D, k, M) is a No instance, or obtains a zone decomposition $V(D) = M \uplus R \uplus (\biguplus_{Z \in \mathcal{Z}} Z)$ with the properties in Lemma 5.3.1. In this section, we fix an arbitrary zone $Z \in \mathcal{Z}$ and give

a polynomial time algorithm (Lemma 5.4.1) to mark a small set of vertices in Z which in some sense "represents" all partial solutions in Z. Such a set will then be used to design reduction rules that bound the degree of the vertices in Z that are not in the representative, which will further be useful to decompose Z into a small number of protrusions. We now formally define the desired set.

Definition 5.4.1 (k-DFVS Representative in Z)**.** *For a digraph D, $Z \subseteq V(D)$ and an integer k, we say that $\Gamma_{\text{DFVS}} \subseteq Z$ is a k-*DFVS representative in *Z if the following holds. If D has a directed feedback vertex set of size at most k, then it also have a directed feedback vertex set S of size at most k where $S \cap Z \subseteq \Gamma_{\text{DFVS}}$.*

Lemma 5.4.1 (k-DFVS Representative Marking Lemma)**.** *There is an algorithm that given a digraph D, $Z \subseteq V(D)$ and an integer k, runs in time $2^{\mathcal{O}(\eta^2)} \cdot (k\ell)^{\mathcal{O}(\eta^2)} \cdot (n+m)$, and returns a set $\Gamma_{\text{DFVS}} \subseteq Z$ of size $(k\ell)^{\mathcal{O}(\eta^2)}$ such that Γ_{DFVS} is a k-DFVS representative in Z.*[1]

We prove Lemma 5.4.1 in two parts. In Section 5.4.1, we revisit the relation between DFVS and Skew Multicut (defined later). Using this relation, we conclude that Γ_{DFVS} can be computed by taking the union of skew multicuts of "appropriate" instances of Skew Multicut. The problem at this stage stems from the fact that the set of appropriate instances that we need to consider is not polynomially bounded, and hence a naive approach of finding a solution to each of these instances and taking their union does not work. This issue is tackled in Section 5.4.2.

5.4.1 Revisiting the Relation with the Skew Multicut Problem

Towards the definition of Skew Multicut, we first define a skew multicut in a digraph. Let D be a digraph, and $\mathcal{P} = ((s_1, t_1), \ldots, (s_p, t_p))$ be an *ordered* set of pairs of vertices (called *terminals*) of D. A *skew multicut* in D with respect to $\mathcal{P}$ is a set S of non-terminal vertices of D such that for all $i, j \in \{1, \ldots, p\}$ with $i \leq j$, there is no path from s_i to t_j in $D - S$. [2] In Skew Multicut (SMC), the input is a digraph D, an ordered set $\mathcal{P} = ((s_1, t_1), \ldots, (s_p, t_p))$ and an integer k. The goal is to decide whether D has a skew multicut of size at most k with respect to $\mathcal{P}$. In order to state the relation between our problem and Skew Multicut, the following

[1] Throughout the chapter, we do not hide constants that depend on η in the $\mathcal{O}$ notation.

[2] In the standard definition of a skew multicut, the latter condition is replaced by the following symmetric condition: For all $i, j \in \{1, \ldots, p\}$ with $j \leq i$, there is no path from s_i to t_j in $D - S$. Here, it is convenient to use $i \leq j$ rather than $j \leq i$.

notation will come handy. Informally, for a digraph D and a subset $B \subseteq V(D)$, we "split" each vertex in B into two distinct vertices, and thereby define the digraph $D_{\dagger B}$. The two fractions of each split vertex will latter correspond to a terminal pair. Formally, we construct $D_{\dagger B}$ as follows.

Definition 5.4.2. *Let D be a digraph and $B \subseteq V(D)$. The digraph $D_{\dagger B}$ is obtained from D as follows. Replace each vertex $v \in B$ by two new vertices v^{out} and v^{in}, add the arc (v^{in}, v^{out}) and replace each arc $(u, v) \in E(D)$ by (u, v^{in}) and each arc $(v, u) \in E(D)$ by (v^{out}, u).*

Lemmas 5.4.2 and 5.4.3 show that any DFVS solution restricted to Z is a skew multicut solution to an appropriate instance of Skew Multicut and vice versa.

Lemma 5.4.2. *Let D be a digraph, $Z \subseteq V(D)$ and $k \in \mathbb{Z}$. Let S be a directed feedback vertex set in D of size at most k. Let $B = N(Z) \setminus S$, and let $\pi_B = (v_1, \ldots, v_b)$ be an ordering of B induced by a topological ordering π of $D - S$. Denote $D' = D[Z \cup B]_{\dagger B}$, $\mathcal{P} = ((v_1^{out}, v_1^{in}), \ldots, (v_b^{out}, v_b^{in}))$ and $k' = |S \cap Z|$. Then, there is a skew multicut in D' with respect to $\mathcal{P}$ of size at most k', that is, $(D', \mathcal{P}, k')$ is a* Yes *instance of* Skew Multicut.

Proof. We claim that $S' = S \cap Z$ is a solution to the Skew Multicut instance $(D', \mathcal{P}, k')$. Observe that the intersection of S' with the set of terminals in $\mathcal{P}$ is empty. Clearly, $|S'| \leq k'$. Suppose, for the sake of contradiction, that S' is not a skew multicut in D' with respect to $\mathcal{P}$. Then, there exist v_i^{out} and v_j^{in}, $i < j$, such that there is a path from v_i^{out} to v_j^{in} in $D' - S'$. From the construction of D', there is a path from v_i to v_j in $D - S$. This is a contradiction because π_B is induced by some topological ordering of $D - S$, and hence there cannot be a path from v_i to v_j, $i < j$, in $D - S$. □

Lemma 5.4.3. *Let D be a digraph, $Z \subseteq V(D)$ and $k \in \mathbb{Z}$. Let S be a directed feedback vertex set in D of size at most k. Let $B = N(Z) \setminus S$ and let $\pi_B = (v_1, \ldots, v_b)$ be an ordering of the vertices of B induced by a topological ordering π of $D - S$. Denote $D' = D[Z \cup B]_{\dagger B}$, and $\mathcal{P} = ((v_1^{out}, v_1^{in}), \ldots, (v_b^{out}, v_b^{in}))$. Let S' be any skew multicut in D' with respect to $\mathcal{P}$. Then, $S^* = (S \setminus Z) \cup S'$ is a directed feedback vertex set in D.*

Proof. Suppose, for the sake of contradiction, that S^* is not a directed feedback vertex set in D. Then, there exists a directed cycle C in $D - S^*$. Since $D[Z]$ is acyclic (from Lemma 5.3.1) and $S^* \setminus Z = S \setminus Z$, we have that $C \cap B \neq \emptyset$ and

$C \cap Z \neq \emptyset$. Let us first consider the case where $|C \cap B| = 1$. In this case, let $C \cap B = \{v_i\}$. Since C is a cycle in $D - S^*$ and $S^* \cap Z = S'$, we observe (from the construction of D') that there is path from v_i^{out} to v_i^{in} in $D' - S'$. This is a contradiction to the assumption that S' is a skew multicut in D' with respect to $\mathcal{P}$. Henceforth, we assume that $|C \cap B| \geq 2$.

Let P be some directed path of the cycle C such that the endpoints of P are in $B = \{v_1, \ldots, v_b\}$ and all its internal vertices are in Z. Let the first vertex of P be v_i and the last vertex of P be v_j. Since S' is a skew multicut solution of the instance stated in the lemma, and because $S' \subseteq S^*$, we have that $j < i$. Since P is a path of the cycle C, there exists another path P' from v_j to v_i in $D - S^*$, such that the union of the sets of arcs of P and P' forms the cycle C. We will show that the existence of the path P' from v_j to v_i in $D - S^*$ (with $j < i$) implies that there exists a path P'' from some v_q to some v_r with $q < r$, such that either *(i)* all the internal vertices of P'' belong to $Z \setminus S^*$ (which includes the case where P'' has no internal vertices), or *(ii)* all the internal vertices of P'' belong to $(V(D) \setminus Z) \setminus S^*$. Let us first show why, to complete the proof, it suffices to prove the existence of such a path P''. In the first case, from the construction of D' and because $S^* \cap Z = S'$, we have that there is a path from v_q^{out} to v_r^{in} in $D' - S'$. Such a path cannot exist because S' is a skew multicut solution in D' with respect to $\mathcal{P}$. In the second case, since the ordering $\{v_1, \ldots, v_b\}$ is induced by some topological ordering of $D - S$ and $S^* \setminus Z = S \setminus Z$, the path P'' again should not exist. Thus, both cases lead to a contradiction.

Let us now describe how to obtain the path P'' from P'. Recall that P' is a path from v_j to v_i (with $j < i$) in $D - S^*$. Let v_r be the first internal vertex of P' such that $v_r \in B$ and $j < r$ (such a vertex exists because $v_i \in B$ and $j < i$). Now consider this sub-path of P' from v_j to v_r. Let v_q be the last vertex on this path that belongs to B and which is not equal to v_r (q could be equal to j). Now, consider the subpath of P' from v_q to v_r. We claim that this subpath is the desired path P''. Since v_q is the last vertex that belongs to B on the path from v_j to v_r that is not v_r itself, the set of internal vertices of P'' has an empty intersection with B. If there are at least two internal vertices in P'', say x, y, such that $x \in Z \setminus S^*$, $y \in (V(D) \setminus Z) \setminus S^*$ and there is a path from x to y in P'', then this path has to pass through a vertex of B (because $N(Z) \setminus S = B$ and $S^* \setminus Z = S \setminus Z$). Thus, we have reached a contradiction. The other case, where $x \in (V(D) \setminus Z) \setminus S^*$ and $y \in Z \setminus S^*$, is symmetric. This shows that P'' must satisfy one of the conditions *(i)* and *(ii)*, and hence the proof is complete. □

The number of guesses for the Skew Multicut instance for which the in-

tersection of a potential DFVS solution with Z is a skew multicut solution is $2^{|N(Z)|} \cdot |N(Z)|^{|N(Z)|}$. This is not polynomially bounded in k and ℓ. In the next section, we see how to compute a set containing some skew multicut solution to each of these instances without having to go over the instances individually.

5.4.2 Computing Solutions for All Instances of Skew Multicut

We formalize the notion of "all possible choices for the appropriate instance of Skew Multicut", by defining a family of instances of Skew Multicut denoted by $\mathcal{F}_{\text{SMC}}$. To simplify notation, for a digraph D and a (not necessarily ordered) set $\mathcal{P}$ of terminal pairs, let $D - \mathcal{P}$ be the digraph obtained from D by deleting all terminals in $\mathcal{P}$. Similarly, for a subset $X \subseteq V(D)$, let $X - \mathcal{P}$ be the set of vertices obtained from X by deleting all terminals in $\mathcal{P}$. To improve readability, *unordered* sets of terminal pairs will be denoted by $\mathcal{Q}$ rather than $\mathcal{P}$. We also stress that in what follows, k should be thought of as a small constant, because here it does not refer to the original k in the input instance of DFVS, but to the parameter set up when we construct an instance of Skew Multicut.

Definition 5.4.3. *Given a digraph D, an* unordered *set $\mathcal{Q} = \{(s_i, t_i) : i \in \{1, \ldots, p\}, s_i, t_i \in V(D)\}$ and an integer k, $\mathcal{F}_{\text{SMC}}(D, \mathcal{Q}, k)$ is a family of instances of* Skew Multicut *such that for each $P^* \subseteq \{1, \ldots, p\}$, for each ordering π of P^* and for each $k' \leq k$, the instance $(D - (\mathcal{Q} \setminus \mathcal{P}^*), \mathcal{P}^*, k')$ belongs to $\mathcal{F}_{\text{SMC}}(D, \mathcal{Q}, k)$ where $\mathcal{P}^* = ((s_{\pi(i)}, t_{\pi(i)}) : i \in P^*)$.*

We clarify that the above notation $\mathcal{P}^* = ((s_{\pi(i)}, t_{\pi(i)}) : i \in P^*)$ means that for all $i, j \in P^*$, we have that $(s_{\pi(i)}, t_{\pi(i)})$ is ordered before $(s_{\pi(j)}, t_{\pi(j)})$ if and only if $i < j$. Similar to the notion of a k-DFVS representative of Z, we first define the notion of a k-SMC representative. The construction of a set that, for any instance in $\mathcal{F}_{\text{SMC}}$, contains some solution for that instance, is captured by the Lemma 5.4.4.

Definition 5.4.4 (k-SMC Representative)**.** *Given a digraph D, a set $\mathcal{Q} = \{(s_i, t_i) : i \in \{1, \ldots, p\}, s_i, t_i \in V(D)\}$ and an integer k, a k-SMC representative in D with respect to $\mathcal{Q}$ is a subset $\Gamma_{\text{SMC}} \subseteq D$ such that each* Yes *instance in the family $\mathcal{F}_{\text{SMC}}(D, \mathcal{Q}, k)$ has a solution that belongs to Γ_{SMC}.*

Lemma 5.4.4 (k-SMC Representative Marking Lemma)**.** *There is an algorithm that, given a digraph D $p, k \in \mathbb{N}$ and $\mathcal{Q} = \{(s_i, t_i) : i \in \{1, \ldots, p\}, s_i, t_i \in V(D)\}$,*

runs in time $p^{\mathcal{O}(k^2)} \cdot (n+m)$ *time and outputs a* k*-SMC representative in* D *with respect to* $\mathcal{Q}$ *of size at most* $k^2(k+1)^k \cdot p^{k(k+2)} \cdot 4^{k^2}$.

The rest of this section concerns the proof of Lemma 5.4.4. We first explain (intuitively) how the algorithm of Lemma 5.4.4 works. Since $|\mathcal{F}_{\text{SMC}}(D, \mathcal{Q}, k)|$ is exponential in p, if a k-SMC representative in D with respect to $\mathcal{Q}$, say Γ_{SMC}, of the desired size exists, then the solutions of "many" instances in $\mathcal{F}_{\text{SMC}}(D, \mathcal{Q}, k)$ intersect "a lot". This is exactly what we want to exploit. Roughly speaking, we want to recursively apply the following step. In each recursive call, partition the instances of $\mathcal{F}_{\text{SMC}}(D, \mathcal{Q}, k)$ into $p^{\mathcal{O}(k)}$ classes, and for each class find a set that is guaranteed to be contained in some solution for each of the instances in the class. Delete this set from the instances in the class, and recurs. Note that we keep track of deleted vertices, since they are precisely the vertices that will form Γ_{SMC}. Since we are looking for a k-sized solution in each instance in $\mathcal{F}_{\text{SMC}}(D, \mathcal{P}, k)$, the depth of the recursion is at most k and hence, we can form the set Γ_{SMC} of the desired size (i.e., $\mathcal{O}(p^{g(k)})$ for some function g of k).

Now, we try to formalize this approach; depending on the obstacles faced, we add layers and machinery to the outline above. First consider the step of partitioning the instances of $\mathcal{F}_{\text{SMC}}(D, \mathcal{Q}, k)$ into some $p^{\mathcal{O}(k)}$ classes, such that for each class there is a set guaranteed to be contained in some solution for each of the instances in the class. To this end, we first try the power of the Pushing Lemma for Skew Multicut, defined below. Roughly speaking, this lemma states that any Yes instance $(D, ((s_i, t_i) : i \in \{1, \ldots, a\}), k)$ of Skew Multicut (for instances in $\mathcal{F}_{\text{SMC}}(D, \mathcal{Q}, k)$, we have $a \leq p$) has a solution of size at most k that contains an important $(\{s_1\}, \{t_1, \ldots, t_a\})$-separator of size at most k.

Proposition 5.4.1 (Pushing Lemma for Skew Multicut, [49])**.** *For a* Yes*-instance* $(D, \mathcal{P} = ((s_1, t_1), \ldots, (s_p, t_p)), k)$ *of* Skew Multicut*, there is a solution* S^* *containing an important* $(\{s_1\}, \{t_1, \ldots, t_p\})$*-separator in* D *of size at most* k*.*

Now, consider any $P^* \subseteq \{1, \ldots, p\}$. Assume w.l.o.g. that $P^* = \{1, \ldots, p^*\}$. Consider all Yes instances in $\mathcal{F}_{\text{SMC}}(D, \mathcal{P}, k)$ where the first terminal pair is (s_1, t_1) and the other terminal pairs are $\{(s_2, t_2), \ldots, (s_{p^*}, t_{p^*})\}$ in some order. Then, by Proposition 5.4.1, for each of these instances there exists a solution containing some important $(\{s_1\}, \{t_1, \ldots, t_{p^*}\})$-separator of size at most k. This is not exactly what we wanted (since we do not obtain a *single* set that is contained in some solution for each of the instances), but we can still work with this as the number of important separators of size at most k is at most 4^k (from Proposition 2.4.3). Then, we can

branch on which important separator to add to Γ_{SMC}. Thus, Proposition 5.4.1 seems to give a way to go about constructing Γ_{SMC}.

However, we are not done yet because if we naively utilize the Pushing Lemma approach, we need to partition $\mathcal{F}_{\text{SMC}}(D, \mathcal{P}, k)$ into $2^p \cdot p$ classes. Indeed, we have 2^p possibilities to choose a subset $P^* \subseteq \{1, \ldots, p\}$ (which captures the indices of the terminals pairs in $\mathcal{P}$ that should not be deleted), and $p^* \leq p$ choices for which is the index in P^* of the first terminal pair (from which we push our solution as described above). For us, $2^p \cdot p$ is a huge number. To handle this issue, we introduce another tool, called the *Important Separator Preservation (via Small Sink Set) Lemma* (formally defined later). Intuitively, this lemma says that if I is an important (X, Y)-separator, then I is also an important (X, Y')-separator for some $Y' \subseteq Y$ where the size of Y' is at most the size of I plus one.

Lemma 5.4.5 (Important Separator Preservation (via Small Sink Set) Lemma)**.** *Let D be a digraph with $X, Y \subseteq V(D)$ and an important (X, Y)-separator $S \subseteq V(D)$ of size α. There is $Y' \subseteq Y$ of size $\alpha + 1$ such that S is an important (X, Y')-separator in $D - (Y \setminus Y')$.*

Proof. Let R be the set of vertices reachable from X in $D - S$, and denote $R_S = R \cup S$. Let $S^* \subseteq V(D)$ be such that $S^* \cap R_S \subsetneq S$, $S^* \cap Y = \emptyset$ and there is no path from $R_S \setminus S^*$ to Y in $D - S^*$. Since S is an important (X, Y)-separator of size α and $R \subsetneq R_S$, any such S^* has size at least $\alpha + 1$. From Menger's Theorem, there are at least $\alpha + 1$ internally vertex disjoint paths from $R_S \setminus S^*$ to Y, all of whose internal vertices are in $D - ((R_S \setminus S^*) \cup Y)$. Consider an arbitrary collection of exactly $\alpha + 1$ of these paths. Let Y' be the subset of Y which contains the endpoints in Y of the $\alpha + 1$ paths in this collection. Observe that these $\alpha + 1$ paths exist even in $D - (Y \setminus Y')$. Since there are $\alpha + 1$ internally vertex disjoint paths from $R_S \setminus S^*$ to Y' in $D - (Y \setminus Y')$, from Menger's Theorem, any set S^* with $S^* \cap R_S \subsetneq S$ and $S^* \cap Y' = \emptyset$, that kills all paths from $R_S \setminus S^*$ to Y' in $D - (Y \setminus Y')$, is of size $\alpha + 1$.

Suppose, for the sake of contradiction, that S is not an important (X, Y')-separator in $D - (Y \setminus Y')$. Then there exists S' such that *(i)* $|S'| \leq |S| = \alpha$, *(ii)* S' is an (X, Y')-separator in $D - (Y \setminus Y')$, and *(iii)* $R \subsetneq R'$ (where R is the set of vertices reachable from X in $D - S$ and R' is the set of vertices reachable from X in $D - S'$). Note that $S' \cap R_S \subsetneq S$ (because $R \subsetneq R'$ and $S \neq S'$) and $S' \cap Y' = \emptyset$. Also, S' kills all paths from $R_S \setminus S'$ to Y' in $D - (Y \setminus Y')$. But this is a contradiction because we have already proved that any such S' has size at least $\alpha + 1$. $\square$

The observation that we can exploit this lemma in our setting is a crucial insight in the design of our kernel. Recall that by Proposition 5.4.1, we can conclude that for some class of instances, the following property holds: There exists a pair (X, Y), where $X = \{s_i\}$ and Y is some set of terminals t_j, such that there is an important (X, Y)-separator of size at most k that is contained in some solution for each of the instances in the class. Since the number of important (X, Y)-separators of size at most k is small, we could branch on them. Basically, we combine Lemma 5.4.5 with Proposition 5.4.1 to add another layer of branching. Below, we briefly discuss the meaning of this extra layer.

Here, we partition our instances into p classes: All instances that have (s_i, t_i) as the first terminal pair (recall that the set of terminal pairs in Skew Multicut is ordered) belong to the same class. While before we had a refined partition with $2^p \cdot p$ classes, here we only have p classes, but which *at first glance* seem non-informative. However, we show that (by Lemma 5.4.5) not much additional information is needed. More precisely, we argue that for all Yes instances in the same class of our rough partition, there exists some $p^{\mathcal{O}(k)}$ sized collection of pairs $\{(X_i, Y_i) : i \in p^{\mathcal{O}(k)}\}$ with the following property: For any instance in the class, there exists a pair in this collection, say (X_i, Y_i), such that there exists an important (X_i, Y_i)-separator of size at most k that is contained in some solution of that instance. Since the size of the collection is $p^{\mathcal{O}(k)}$, and for each pair in it there are at most 4^k important separators of size at most k, we branch into at most $p^{\mathcal{O}(k)} \cdot 4^k$ branches for each class. Since $p^{\mathcal{O}(k)} \cdot 4^k$ is small enough to obtain a kernel—recall that in Skew Multicut, k is small (constant) but p is large—let us move ahead to see how we obtain the collection $\{(X_i, Y_i) : i \in p^{\mathcal{O}(k)}\}$.

We claim that for any class, whose first terminal pair is some (s_i, t_i), the collection $\{(s_i, T) : T \subseteq \{t_1, \ldots, t_p\}, |T| \leq k + 1\}$ is precisely that collection that we want. To see this, consider any Yes instance $(D, \mathcal{P}^*, k)$ whose first terminal pair is (s_i, t_i). Let P^* denote the set of indices of the pairs in $\mathcal{P}^*$. By the Pushing Lemma for Skew Multicut, there exists a solution to this instance that contains some important $(\{s_i\}, \{t_j \mid j \in P^*\})$-separator of size at most k. In turn, by the Important Separator Preservation Lemma, there exists $T \subseteq \{t_j \mid j \in P^*\}$ of size at most $k + 1$, such that any important $(\{s_i\}, \{t_j \mid j \in P^*\})$-separator is also an important $(\{s_i\}, T)$-separator! Thus, we conclude that for each Yes instance in $\mathcal{F}_{\text{SMC}}(D, \mathcal{P}, k)$ whose first terminal pair is (s_i, t_i), there exists a pair in the collection $\{(s_i, T) : T \subseteq \{t_1, \ldots, t_p\}\}$ such that one of the important separators of size at most k of this pair is contained in some solution for this instance.

We formalize the approach above in the proof of Lemma 5.4.4 below. Having this lemma at hand, we give a short proof for Lemma 5.4.1 .

5.4.2.1 Proof of the k-SMC Representative Marking Lemma

In this section, we prove Lemma 5.4.4. We start by describing the proposed algorithm for Lemma 5.4.4.

Description of the Algorithm. Our algorithm is a branching (recursive) algorithm. (Here, by branching tree we refer to the tree whose root is the initial call to the algorithm, and which described the relationships between the calls to the algorithm as excepted, that is, the children of a node correspond to the recursive calls made by that node.) Each node of the branching tree corresponds to a triple $(D, \mathcal{Q}, k)$ where D is a directed graph, $\mathcal{Q}$ is an *unordered* set of vertex pairs in D, and k is an integer. The *measure* to analyze the size of the branching tree is k. The algorithm initializes $\Gamma_{\text{SMC}}= \emptyset$ and updates Γ_{SMC} at the end of each branch. The base case occurs when $k \leq 0$. Whenever $k > 0$, the algorithm branches into the branches described below and updates Γ_{SMC}.

Consider a node of the branching tree labelled by $(D, \mathcal{Q}, k)$. We fix some notation before we proceed further. Let $\mathcal{Q} = \{(s_i, t_i) : i \in \{1, \ldots, p\}\}$. By T we denote the set $T = \{t_i : i \in \{1, \ldots, p\}\}$. For any $P' \subseteq \{1, \ldots, p\}$, by $T_{P'}$ we denote the set $T_{P'} = \{t_i : i \in P'\}$, and by $\mathcal{Q}_{P'}$ we denote $\mathcal{Q}_{P'} = \{(s_i, t_i) : i \in P'\}$. We will now describe the branches of the algorithm at the node $(D, \mathcal{Q}, k)$ of the branching tree. For each $i \in \{1, \ldots, p\}$, for each $P' \subseteq \{1, \ldots, p\}$ of size at most $k + 1$ and for each $I \neq \emptyset$ which is an important $(s_i, T_{P'})$-separator in $D - (\mathcal{Q} \setminus \mathcal{Q}_{P' \cup \{i\}})$ of size at most k, there is a branch that corresponds to the triple (i, P', I). A branch (i, P', I) at a node $(D, \mathcal{Q}, k)$ results in a node that corresponds to the triple $(D - I, \mathcal{Q} \setminus \{(s_i, t_i)\}, k - |I|)$. Also, at the end of this branch, Γ_{SMC} is updated as $\Gamma_{\text{SMC}} = \Gamma_{\text{SMC}} \cup I$. Observe that the measure in each branch drops by at least 1 because $I \neq \emptyset$.

The size of the branching tree. Since the number of important separators of size at most k is at most 4^k (Proposition 2.4.3), observe that at any node of the branching tree, the number of branches, $w \leq p \cdot \sum_{i=0}^{k+1} \binom{p}{i} \cdot 4^k \leq (k + 1) \cdot p^{k+2} \cdot 4^k$. Since the measure of the branching tree is k, the branching algorithm halts when $k \leq 0$ and in each branch the measure strictly decreases, the depth of the branching tree is at most k. Thus, the number of nodes in the branching tree is at most $\sum_{i \in \{1, \ldots, k\}} w^i \leq k \cdot w^k \leq k(k+1)^k \cdot p^{k(k+2)} \cdot 4^{k^2}$. Since, at each node of the branching tree, the size of Γ_{SMC} increases by at most k, $|\Gamma_{\text{SMC}}| = k^2(k + 1)^k \cdot p^{k(k+2)} \cdot 4^{k^2}$.

Running Time. At any node $(D, \mathcal{Q}, k)$, for each $i \in \{1, \ldots, p\}$ and $P' \subseteq \{1, \ldots, p\}$ of size at most $k+1$, we need to compute the set of appropriate important separators of size at most k. From Proposition 2.4.3, this takes time, say $t = p \cdot \sum_{i=0}^{k+1} \binom{p}{i} \cdot \mathcal{O}(4^k \cdot k^2 \cdot (n+m)) = \mathcal{O}(p^{k+2} \cdot 4^k \cdot k^3 \cdot (n+m))$. Recall that at any node of the branching tree, the number of branches is $w \leq (k+1) \cdot p^{k+2} \cdot 4^k$. Let $T(k)$ be the time taken by the algorithm when the input triple $(D, \mathcal{Q}, k)$ has measure k. Then, we have the following recurrence relation: if $k \leq 0$, $T(k) = 1$; otherwise, $T(k) \leq wT(k-1) + t$. Solving this recurrence, we get that $T(k) \leq w^k \cdot t = \mathcal{O}(p^{(k+1)(k+2)} \cdot 4^{k(k+1)} \cdot (k+1)^k \cdot k^3 \cdot (n+m))$.

Correctness. Suppose that the algorithm takes as initial input the instance $(D, \mathcal{Q}, k)$. Consider an instance $(D, \mathcal{P}^*, k^*)$ in $\mathcal{F}_{\text{SMC}}(D, \mathcal{P}, k)$ which is a YES instance of SKEW MULTICUT. Without loss of generality, let $\mathcal{P}^* = ((s_i, t_i) : i \in \{1, \ldots, p^*\})$. We need to show that there exists a skew multicut solution of the instance $(D, \mathcal{P}^*, k^*)$ contained inside Γ_{SMC}. Let μ be the depth of the algorithm. For any $d \leq \mu$, let $\Gamma_{\text{SMC}d}$ be the partial set (subset of Γ_{SMC}) constructed by the algorithm at the end of the last branch at depth d of its branching tree. We will now prove the following claim by induction on d, and then later show how the correctness of the algorithm follows from this claim.

Claim 5.4.1. *For any $d \leq \mu$, there exists $S_d \subseteq \Gamma_{\text{SMC}d}$ with the following properties.*

1. *There is a solution S to the instance $(D, \mathcal{P}^*, k^*)$ of* SKEW MULTICUT *such that $S_d \subseteq S$.*
2. *For all $i \in \{1, \ldots, d\}$, $j \in \{1, \ldots, p^*\}$ with $j \geq i$, there is no path from s_i to t_j in $D - S_d$.*
3. *There is a path from the root of the branching tree to one of its nodes at depth d such that S_d is the union of the sets added to Γ_{SMC} at each node along this path.*

Proof. We will prove this claim by induction on d, that is, the depth of the branching tree.

Base Case. Suppose $d = 1$. Since $(D, \mathcal{P}^*, k^*)$ is a YES instance of SKEW MULTICUT, from Proposition 5.4.1, there exists an important $(s_1, \{t_1, \ldots, t_{p^*}\})$-separator, say I, of size at most k^* $(\leq k)$ in D such that there is no path in $D - I$ from s_1 to t_j, for all $j \in P^*$, and there exists a solution S to the instance $(D, \mathcal{P}^*, k^*)$ such that $I \subseteq S$. We will now prove that $I \subseteq \Gamma_{\text{SMC}1}$. From Lemma 5.4.5, there exists

$T \subseteq \{t_1, \ldots, t_{p^*}\}$ such that $|T| \leq k+1$ and I is an important (s_i, T)-separator in $D - (\mathcal{P}^* \setminus \{(s_i, t_i) : t_i \in T\})$ of size at most k. Consider the branch that corresponds to the triple $(i, \{j : t_j \in T\}, I)$. At the end of this branch, Γ_{SMC} is updated as $\Gamma_{\text{SMC}} = \Gamma_{\text{SMC}} \cup I$. This proves the base case.

Inductive Step: Suppose that the claim holds for $d-1$. By the induction hypothesis, there exists a set $S_{d-1} \subseteq \Gamma_{\text{SMC}d-1}$ with the properties mentioned in Claim 5.4.1. Let $(D', \mathcal{Q}', k')$ be the triple corresponding to the node at level $d-1$ which is the end-point of the path in the branching tree along which S_{d-1} is constructed. Observe that $\mathcal{Q}' = \{(s_i, t_i) : i \in \{1, \ldots, p^*\}, i \geq d\}$, $D' = D - S_{d-1}$ and $k' = k - |S_{d-1}|$. We now prove the three properties stated in the claim.

Property 1. Let S' be a solution to the Skew Multicut instance $(D', ((s_i, t_i) : i \in \{1, \ldots, p^*\}, i \geq d), k')$. We first claim that $S_{d-1} \cup S'$ is a solution to the Skew Multicut instance $(D, \mathcal{P}^*, k^*)$. From the induction hypothesis, for all $i \in \{1, \ldots, d-1\}$, $j \in \{1, \ldots, p^*\}$ with $j \geq i$, there is no path from s_i to t_j in $D - S_{d-1}$. Since S' is a skew multicut solution to $(D', ((s_i, t_i) : i \in \{1, \ldots, p^*\}, i \geq d), k')$ and $D' = D - S_{d-1}$, for all $i, j \in \{1, \ldots, p^*\}$ with $j \geq i$, there is no path from s_i to t_j in $D - (S_{d-1} \cup S')$. We now need to show that $|S_{d-1} \cup S'| \leq k^*$, that is, $|S_{d-1}| + k' \leq k^*$. To prove this, note that, from the induction hypothesis, there exists a solution, say S, of $(D, \mathcal{P}^*, k^*)$, such that $S_{d_1} \subseteq S$. Since $k' = k - |S_{d-1}| \geq k^* - |S_{d-1}|$, we conclude that $k^* \geq k' + |S_{d-1}|$.

Thus, we have proved that there exists a solution, say S, to the Skew Multicut instance $(D, \mathcal{P}^*, k^*)$ such that $S = S_{d-1} \cup S'$, where S' is any solution to the Skew Multicut instance $(D', ((s_i, t_i) : i \in \{1, \ldots, p^*\}, i \geq d), k')$. From Proposition 5.4.1, we know that there exists a skew multicut solution S' for $(D', ((s_i, t_i) : i \in \{1, \ldots, p^*\}, i \geq d), k')$ which contains an important $(s_i, \{t_i : i \in \{1, \ldots, p^*\}, i \geq d\})$-separator, say I, in D'. Let S' be such a solution to $(D', ((s_i, t_i) : i \in \{1, \ldots, p^*\}, i \geq d), k')$. From Lemma 5.4.5, there exists $T \subseteq \{t_d \ldots, t_{p^*}\}$ of size at most $k+1$ such that I is an important (s_d, T)-separator in $D' - (\mathcal{Q}' \setminus \{(s_i, t_i) : i \in T\})$ of size at most k. Consider the branch at $(D', \mathcal{Q}', k')$ that corresponds to the triple $(d, \{(s_i, t_i) : i \in T\}, I)$. At the end of this branch, I is added to Γ_{SMC}. Let $S_d = S_{d-1} \cup I$.

Since $S_d \cup S'$ is a solution to the Skew Multicut instance $(D, \mathcal{P}^*, k^*)$, where S' is any solution of the Skew Multicut instance $(D', ((s_i, t_i) : i \in \{1, \ldots, p^*\}, i \geq d), k')$, and there exists a solution S' to the Skew Multicut instance $(D', ((s_i, t_i) : i \in \{1, \ldots, p^*\}, i \geq d), k')$ such that $I \subseteq S'$, we conclude that there exists a solution

S to the Skew Multicut instance $(D, \mathcal{P}^*, k^*)$, such that $S_{d-1} \cup I \subseteq S$. Since $S_d = S_{d-1} \cup I$, we conclude that $S_d \subseteq S$.

Property 2. Since, for any $i \in \{1, \ldots, d-1\}$ and for all $j \in \{1, \ldots, p^*\}$ with $j \geq d-1$ there are no paths from s_i to t_j in $D - S_{d-1}$ (from induction hypothesis), and I is an $(s_d, \{t_d, \ldots, t_{p^*}\})$-separator in $D - S_{d-1}$, we have that for all $i \in \{1, \ldots, d\}$ and for all $j \in \{1, \ldots, p^*\}$ with $j \geq d$, there is no path from s_i to t_j in $D - S_d$.

Property 3. Consider the path in the branching tree along which S_{d-1} is constructed. Such a path exists because of induction hypothesis. Since $(D', \mathcal{Q}', k')$ is the last node on this path, the set I is added to Γ_{SMC} at one of the branches from $(D', \mathcal{Q}', k')$, we have that S_d is the union of the sets added to Γ_{SMC} along a path in the branching tree. This concludes the proof of the claim. □

If μ is the depth of the branching tree of the algorithm, consider the set S_μ as defined in Claim 5.4.1. Consider the path in the branching tree that corresponds to the construction of S_μ. Since μ is the depth of the branching tree of the algorithm and the algorithm terminates only when $k^* \leq 0$, $|S_\mu| \geq k^*$. From Claim 5.4.1, there exists a solution S to the instance $(D^*, \mathcal{P}^*, k^*)$ such that $S_\mu \subseteq S$. Since $|S| \leq k^*$, $|S_\mu| \geq k^*$ and $S_\mu \subseteq S$, we have that $S_\mu = S$. Since $S_\mu \subseteq \Gamma_{\text{SMC}}$, we have that $S \subseteq \Gamma_{\text{SMC}}$.

5.4.2.2 Proof of the k-DFVS Representative Marking Lemma

Recall that the input for the algorithm is a digraph D, an integer k and a set Z. We run the algorithm of Lemma 5.4.4 on the input $(D[N[Z]]_{\dagger N(Z)}, \{(v^{out}, v^{in}) : v \in N(Z)\}, 2(\eta+1))$. Let Γ_{SMC} be the output of this algorithm. We claim that it is safe to set $\Gamma_{\text{DFVS}} = \Gamma_{\text{SMC}}$, that is, Γ_{SMC} is a k-DFVS representative in Z. First observe that, from Lemma 5.4.4, $|\Gamma_{\text{SMC}}| = |N(Z)|^{\mathcal{O}(\eta^2)} = |M \cup R|^{\mathcal{O}(\eta^2)} = (k \cdot \ell)^{\mathcal{O}(\eta^2)}$. Next, suppose that (D, k, M) is a YESinstance of DFVS/DFVS+Tw-η Mod. We now show that there is a solution S^* such that $S^* \cap Z \subseteq \Gamma_{\text{SMC}}$. Since (D, k, M) is a Yes instance of DFVS/DFVS+Tw-η Mod, from Lemma 5.3.2, there exists a solution S such that $|S \cap Z| \leq 2(\eta+1)$. If $S \cap Z \subseteq \Gamma_{\text{SMC}}$, then $S = S^*$. Otherwise, let $B = N(Z) \setminus S$ and let $\pi_B = \{v_1, \ldots, v_b\}$ be an ordering of the vertices of B induced by some topological ordering of $D - S$. Consider the instance $(D[Z \cup B]_{\dagger B}, \{(v_i^{out}, v_i^{in}) : i \in \{1, \ldots, b\}\}), 2(\eta+1))$ of Skew Multicut. If this is a Yes instance, then there exists $S' \subseteq Z$ such that S' is a solution of the instance $(D[Z \cup B]_{\dagger B}, \{(v_i^{out}, v_i^{in}) : i \in \{1, \ldots, b\}\}), 2(\eta+1))$ and $S' \subseteq \Gamma_{\text{SMC}}$.

Since $|S \cap Z| \leq 2(\eta+1)$, from Lemmas 5.4.2 and 5.4.3, $(D[Z \cup B]_{\dagger B}, \{(v_i^{out}, v_i^{in}) : i \in \{1, \ldots, b\}\}), 2(\eta+1))$ is indeed a YES instance of SKEW MULTICUT and $S^* = (S \setminus Z) \cup S'$ is a solution to the instance (D, k, M) of directed feedback vertex set. Thus, we conclude that $S^* \cap Z \subseteq \Gamma_{\text{SMC}}$.

5.5 Reduction Rules

In this section we give reduction rules to reduce the size of each "zone". More precisely, we first apply the algorithm of Lemma 5.3.1 which either correctly decides that (D, k, M) is a NO instance, or constructs a zone-decomposition $V(D) = M \uplus R \uplus (\biguplus_{Z \in \mathcal{Z}} Z)$ with $|\mathcal{Z}| \leq 6k(\ell^2+1)$ and $|R| \leq 2(\eta+1)k(\ell^2+1)$. For a fixed zone $Z \in \mathcal{Z}$, we concentrate on reducing the size of Z. Once we are able to bound the size of each zone by a polynomial function of k and ℓ, we obtain a polynomial kernel for our problem. Thus, from now onwards we concentrate on bounding the size of a single zone Z.

For the rest of this section, consider the zone-decomposition $V(D) = M \uplus R \uplus (\biguplus_{Z \in \mathcal{Z}} Z)$ computed by the algorithm of Lemma 5.3.1 on input (D, k, M). Let $Z \in \mathcal{Z}$ be an arbitrarily fixed zone and let Γ_{DFVS} be a k-DFVS representative in Z, computed using the algorithm of Lemma 5.4.1. We bound the size of Z in a two step procedure. In the first step, described in Section 5.5.1, we design reduction rules that remove all the arcs between M and $Z \setminus \Gamma_{\text{DFVS}}$ (at the cost of adding arcs between M and Γ_{DFVS}). Once this is done, we have that Z interacts with the "outside world" in a limited fashion via Γ_{DFVS} alone. After we have achieved this, in the second step, described in Section 5.5.2, we will be able to partition $Z \setminus \Gamma_{\text{DFVS}}$ into a "small" number of slices such that each slice has treewidth at most η and has at most $\mathcal{O}(\eta)$ neighbors outside (that is, the slice is an $\mathcal{O}(\eta)$-protrusion). Each such slice can then be replaced by a constant size equivalent slice using the protrusion replacement machinery. We will finally conclude this section with the proof of our main theorem, Theorem 5.0.1.

In the upcoming subsections, we give several reduction rules. These rules are *applied in the given order*, exhaustively. That is, at any point of time we apply the lowest numbered reduction rule that is applicable. In all our reduction rules, we reduce an instance (D, k, M) to (D', k', M'), $M' \subseteq M$ and $k' \leq k$.

5.5.1 Limiting the Interaction Between M and Z

Recall that our goal is to design reduction rules that eventually remove all arcs between M and $Z \setminus \Gamma_{\text{DFVS}}$. This is achieved at the cost of adding arcs among the vertices of M, and between vertices in M and vertices in Γ_{DFVS}. The first set of reduction rules ensures that the addition of this set of arcs is safe. Using this, we then design reduction rules that delete the arcs between M and $Z \setminus \Gamma_{\text{DFVS}}$.

Reduction Rule 5.5.1. *If there exist $u, v \in N(Z) \cap M$ (here, u may be the same vertex as v) such that there is a path from u to v, all of whose internal vertices are in $Z \setminus \Gamma_{\text{DFVS}}$, then add the arc (u, v) to D, if it does not already exist. That is, reduce the instance (D, k, M) to $(D \cup \{(u, v)\}, k, M)$.*

Lemma 5.5.1. *Reduction Rule 5.5.1 is safe.*

Proof. The backward direction is trivial. For ease of notation, let us denote $D \cup \{(u, v)\}$ by D'. For the forward direction, first observe that M is also an η-treewidth modulator in the graph D', that is, $\mathbf{tw}(D' - M) \leq \eta$, because $D' - M = D - M$. If (D, k, M) is a YES instance, then from Lemma 5.4.1, there exists a directed feedback vertex set S in D of size at most k such that $S \cap Z \subseteq \Gamma_{\text{DFVS}}$. We claim that S is also a directed feedback vertex set in D'. Suppose not. Then there is a cycle C in $D' - S$ that uses the arc (u, v). Since there is a path from u to v, all of whose internal vertices are in $Z \setminus \Gamma_{\text{DFVS}}$ and the arc (u, v) is not present in D, there are at least $k + 1$ internally vertex disjoint paths from u to v in D (from Lemma 5.3.1). Since $|S| \leq k$, there is a path from u to v in $D - S$. Replacing the arc (u, v) by this path in C, we get a closed walk in $D - S$, which contradicts the fact that S is a directed feedback vertex set in D. □

Observe that when u and v are the same vertex in Reduction Rule 5.5.1, the resulting digraph will have self-loops. The next reduction rule removes all self-loops in the digraph. It is easy to see that Reduction Rule 5.5.2 is safe.

Reduction Rule 5.5.2. *If there exists $v \in V(D)$ with a self-loop, then delete v and reduce k by 1. That is, reduce the instance (D, k, M) to $(D - \{v\}, k - 1, M \setminus \{v\})$.*

The next reduction rule gives a sufficient condition in which we can add an arc between a vertex in M and a vertex in Γ_{DFVS}.

Reduction Rule 5.5.3. *Suppose that there exist $u \in M$ and $v \in Z \setminus \Gamma_{\text{DFVS}}$ such that $(u, v) \in E(D)$, and $x \in \Gamma_{\text{DFVS}}$ such that there exists a path from v to x, all of*

whose internal vertices are in $Z \setminus \Gamma_{\text{DFVS}}$. *Let* $D' = D \cup \{(u, x)\}$. *Reduce* (D, k, M) *to* (D', k, M).

Lemma 5.5.2. *Reduction Rule 5.5.3 is safe.*

Proof. The backward direction is trivial. For the forward direction, first observe that, M is also a treewidth modulator for the graph D', that is, $\mathbf{tw}(D' - M) \leq \eta$, because $D' - M = D - M$. If (D, k, M) is a YES instance, from Lemma 5.4.1, there exists a directed feedback vertex set S in D of size at most k such that $S \cap Z \subseteq \Gamma_{\text{DFVS}}$. We claim that S is also a directed feedback vertex set in D'. Suppose not. Then there is a cycle C in $D' - S$ that uses the arc (u, x). Consider the path from u to x in D, all of whose internal vertices are in $Z \setminus \Gamma_{\text{DFVS}}$. From the choice of S, such a path exists in $D - S$. Replacing the arc (u, x) in C by this path we get a closed walk in $D - S$, which contradicts the fact that S is a directed feedback vertex set in D. □

Reduction Rule 5.5.4. *Suppose there exist* $u \in M$ *and* $v \in Z \setminus \Gamma_{\text{DFVS}}$ *such that* $(v, u) \in E(D)$, *and* $x \in \Gamma_{\text{DFVS}}$ *such that there is a path from* x *to* v, *all of whose internal vertices are in* $Z \setminus \Gamma_{\text{DFVS}}$. *Let* $D' = D \cup \{(x, u)\}$. *Reduce* (D, k, M) *to* (D', k, M).

The proof of safeness of Reduction Rule 5.5.4 is analogous to the proof of safeness of Reduction Rule 5.5.3 and thus omitted.

The next reduction rules delete arcs between M and $Z \setminus \Gamma_{\text{DFVS}}$.

Reduction Rule 5.5.5. *If there exists* $u \in M$ *and* $v \in Z \setminus \Gamma_{\text{DFVS}}$ *such that* $(u, v) \in E(D)$, *then reduce* (D, k, M) *to* (D', k, M), *where* $D' = D \setminus \{(u, v)\}$.

Lemma 5.5.3. *Reduction Rule 5.5.5 is safe.*

Proof. The forward direction is trivial. For the backward direction, let S be a directed feedback vertex set of D' of size at most k. We claim that S is also a directed feedback vertex set in D. Suppose not. Then there is a cycle C in $D - S$ that uses the arc (u, v). Since C uses the arc (u, v), C passes through Z. Let P be the unique path u to some vertex $w \in N(Z)$, all of whose internal vertices are in C and none in $N(Z)$. Observe that $u \in N(Z)$. If all the internal vertices of P are in $Z \setminus \Gamma_{\text{DFVS}}$, then, since Reduction Rule 5.5.1 has been applied, there is an arc $(u, w) \in E(D)$. Consider the closed walk C' obtained from C after replacing the path P by (u, w). Since C' is a closed walk in $D' - S$, this contradicts that S is a

directed feedback vertex set in D'. In the other case, there exists an internal vertex of P that belongs to Γ_{DFVS}. Let x be the first vertex of P that belongs to Γ_{DFVS}. Since Reduction Rule 5.5.3 has been applied, there exists an arc $(u, x) \in E(D)$ and hence in $E(D' - S)$. Consider the sub-path P' of P from u to x. Replacing P' in C by the arc (u, x), we get a closed walk in $D' - S$, which contradicts that S is a directed feedback vertex set in D'. □

Reduction Rule 5.5.6. *If there exists $u \in M$ and $v \in Z \setminus \Gamma_{\text{DFVS}}$ such that $(v, u) \in E(D)$, then reduce (D, k, M) to (D', k, M), where $D' = D \setminus \{(v, u)\}$.*

The proof of safeness of Reduction Rule 5.5.6 is symmetric to the proof of safeness of Reduction Rule 5.5.5 and thus omitted. Observe that each of the Reduction Rules 5.5.1-5.5.6 can be applied in polynomial time (by using the algorithms for computing shortest path in directed graphs). We conclude this subsection with the following observation.

Observation 5.5.1. *If Reduction Rules 5.5.5 and 5.5.6 are no longer applicable, then $N(Z \setminus \Gamma_{\text{DFVS}}) \cap M = \emptyset$.*

5.5.2 Protrusion Replacement and Proof of the Main Theorem

Recall that the goal now is to slice up $Z \setminus \Gamma_{\text{DFVS}}$ into pieces each of which has treewidth η and $\mathcal{O}(\eta)$ neighbors outside it. Such a slice is what we call an $\mathcal{O}(\eta)$-protrusion. For any positive integer r, the formal definition of an r-protrusion is given below.

Definition 5.5.1 (r-Protrusion)**.** *For any $r \in \mathbb{Z}^+$, an* r-protrusion *in a graph D is a set of vertices $X \subseteq V(D)$ such that $|N(X)| \leq r$ and $\mathbf{tw}(D[X]) \leq r$.*

We need the following lemma which says that if there exists a large enough protrusion, then it can be replaced with an equivalent one of constant size in linear time. As the proof of the lemma requires the introduction of several concepts only relevant to it, we present this proof separately in Section 5.6.

Lemma 5.5.4 (Protrusion Replacer Lemma)**.** *For every $t \in \mathbb{Z}^+$, there exists $c \in \mathbb{Z}^+$ (depending only on t), and an algorithm that, given an instance (D, k, M) of* DFVS/DFVS+Tw-η Mod *and a t-protrusion X in D such that $|X| > c$ and $X \cap M = \emptyset$, outputs, in time $\mathcal{O}(|X|)$, a digraph D' and integer k', such that:*

1. *$|V(D')| < |V(D)|$ and $k' \leq k$,*

2. *D' is obtained from D by deleting some vertices/arcs of X and/or contracting some edges of X, and*

3. *D has a directed feedback vertex set of size at most k if and only if D' has a directed feedback vertex set of size at most k'.*

Observe that the graph D' in Lemma 5.5.4 is a minor of D (for a *directed* graph D, by a minor of D we refers to a minor in the underlying undirected graph of D). In fact, if M is a treewidth-η modulator in D, then M is also a treewidth-η modulator in D'. Note that Lemma 5.5.4 is not replacing a big enough protrusion by any arbitrary smaller protrusion, rather the replacement is such that the resulting graph is a minor of the original graph. This is necessary to ensure that the size of the treewidth-η modulator (a component in the parameter of our problem) does not increase after this replacement.

Henceforth, c is the constant of Lemma 5.5.4 when $t = 4(\eta + 1)$. For a given set $B \subseteq V(D)$, define X_B to be the set of vertices that are either in B or in some connected component of $D - B$ that has treewidth at most η. From Lemma 5.3.1, $\mathbf{tw}(D[Z]) \leq \eta$. Let (T, β) be a nice tree decomposition of $D[Z]$, computed using the algorithms of Propositions 2.5.2 and 2.5.3. Let $B_{\Gamma_{\text{DFVS}}} \subseteq V(T)$ be such that for each $u \in \Gamma_{\text{DFVS}}$, there exists $t \in B_{\Gamma_{\text{DFVS}}}$ such that $u \in \beta(t)$. Observe that $|B_{\Gamma_{\text{DFVS}}}| \leq |\Gamma_{\text{DFVS}}|$. Let $B'_{\Gamma_{\text{DFVS}}}$ be the LCA-closure of $B_{\Gamma_{\text{DFVS}}}$ in T. From Proposition 5.2.1, $|B'_{\Gamma_{\text{DFVS}}}| \leq 2|B_{\Gamma_{\text{DFVS}}}| \leq 2|\Gamma_{\text{DFVS}}|$. Let $\mathcal{C}$ be the collection of connected components of $T - B'_{\Gamma_{\text{DFVS}}}$. Since (T, β) is a nice tree decomposition, for any $t \in V(T)$, the degree of t in T is at most 3. Hence, $|\mathcal{C}| \leq 3 \cdot |B'_{\Gamma_{\text{DFVS}}}| \leq 6|\Gamma_{\text{DFVS}}|$. Let $\widetilde{\Gamma}_{\text{DFVS}} = \bigcup_{t \in B'_{\Gamma_{\text{DFVS}}}} \beta(t)$. Observe that $\Gamma_{\text{DFVS}} \subseteq \widetilde{\Gamma}_{\text{DFVS}}$. For each $C_i \in \mathcal{C}$, define $U_i \subseteq Z \setminus \Gamma_{\text{DFVS}}$ as $U_i = \bigcup_{t \in C_i} \beta(t) \setminus \widetilde{\Gamma}_{\text{DFVS}}$. Since (T, β) is a tree decomposition of width at most η, $|\widetilde{\Gamma}_{\text{DFVS}}| \leq (\eta + 1) \cdot |B'_{\Gamma_{\text{DFVS}}}| \leq 2(\eta + 1) \cdot |\Gamma_{\text{DFVS}}|$. Observe that $Z = \widetilde{\Gamma}_{\text{DFVS}} \uplus \bigcup_{i \in \{1,\ldots,|\mathcal{C}|\}} U_i$. From Proposition 5.2.1, and the edge covering and connectivity properties of a tree decomposition, we have that for each $U_i \in \mathcal{C}$, $|N(U_i) \cap Z| \leq 2(\eta + 1)$. Since $|N(Z) \cap R| \leq 2(\eta + 1)$ (from Lemma 5.3.1), $|N(U_i) \cap R| \leq 2(\eta + 1)$. Furthermore, from Observation 5.5.1, $|N(U_i) \cap M| = 0$. Since $N(Z) \subseteq M \cup R$ (from Lemma 5.3.1) and $U_i \subseteq Z$, we conclude that $|N(U_i)| \leq 4(\eta + 1)$. Also, since $\mathbf{tw}(D[Z]) \leq \eta$ and $U_i \subseteq Z$, we conclude that U_i is a $4(\eta + 1)$-protrusion in D.

Reduction Rule 5.5.7 (Protrusion Replacement Reduction Rule)**.** *If there exists $i \in \{1, \ldots, |\mathcal{C}|\}$, such that $|U_i| > c$, then apply the algorithm of Lemma 5.5.4 on the*

instance (D, k, M) *along with the* $4(\eta+1)$*-protrusion* U_i*. Let* D', k' *be the digraph and integer, respectively, outputted by this algorithm. Then reduce* (D, k, M) *to* (D', k', M)*.*

The safeness of Reduction Rule 5.5.7 follows from Lemma 5.5.4. Also, from the construction of U_i and Lemma 5.5.4, Reduction Rule 5.5.7 can be applied in polynomial time.

Lemma 5.5.5. *When none of the reduction rules is applicable,* $|Z| = (k\ell)^{\mathcal{O}(\eta^2)}$*.*

Proof. Since $Z = \widetilde{\Gamma}_{\text{DFVS}} \uplus \bigcup_{i \in \{1,\ldots,|\mathcal{C}|\}} U_i$, $|\mathcal{C}| \leq 6|\Gamma_{\text{DFVS}}|$, $|\widetilde{\Gamma}_{\text{DFVS}}| \leq 2(\eta+1) \cdot |\Gamma_{\text{DFVS}}|$ and $|U_i| \leq c$ (after the application of Reduction Rule 5.5.7), we have that $|Z| = (k\ell)^{\mathcal{O}(\eta^2)}$ (from Lemma 5.4.1). □

Now that we have bounded the size of each zone Z, we are ready to prove Theorem 5.0.1.

Proof of Theorem 5.0.1. Let (D, k, M) be an instance to DFVS/DFVS+Tw-η Mod. Now we apply Lemma 5.3.1 and either correctly decide that (D, k, M) is a No instance, or construct a zone-decomposition $V(D) = M \uplus R \uplus (\biguplus_{Z \in \mathcal{Z}} Z)$ with $|\mathcal{Z}| \leq 6k(\ell^2+1)$ and $|R| \leq 2(\eta+1)k(\ell^2+1)$. For each $Z \in \mathcal{Z}$, we do the following. We compute a k-DFVS representative in Z and call it Γ_{DFVS}. We then apply all the reduction rules of Section 5.5 exhaustively. From Lemma 5.5.5, when none of the reduction rules are applicable $|Z| = (k\ell)^{\mathcal{O}(\eta^2)}$. Thus, for each $Z \in \mathcal{Z}$, when none of the reduction rules are applicable, $|V(D)| \leq |M| + |R| + (k\ell)^{\mathcal{O}(\eta^2)} \cdot |\mathcal{Z}|$. Thus, from Lemma 5.3.1, $|V(D)| = (k \cdot \ell)^{\mathcal{O}(\eta^2)}$. □

5.6 Towards the Proof of the Protrusion Replacer Lemma

In this section we give a proof of Lemma 5.5.4. We start with some notations and definitions. The next two subsections are taken from [24].

5.6.1 Counting Monadic Second Order Logic and its Properties

The syntax of Monadic Second Order Logic (MSO) of graphs includes the logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, variables for vertices, edges, sets of vertices, and sets of edges, the quantifiers $\forall$, $\exists$ that can be applied to these variables, and the following five binary relations:

1. $u \in U$ where u is a vertex variable and U is a vertex set variable;
2. $d \in D$ where d is an edge variable and D is an edge set variable;
3. $\mathbf{inc}(d, u)$, where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident with the vertex u;
4. $\mathbf{adj}(u, v)$, where u and v are vertex variables and the interpretation is that u and v are adjacent;
5. equality of variables representing vertices, edges, sets of vertices, and sets of edges.

In addition to the usual features of monadic second-order logic, if we have atomic sentences testing whether the cardinality of a set is equal to q modulo r, where q and r are integers such that $0 \leq q < r$ and $r \geq 2$, then this extension of the MSO is called the *counting monadic second-order logic*. Thus CMSO is MSO with the following atomic sentence for a set S:

$\mathbf{card}_{q,r}(S) = \mathbf{true}$ if and only if $|S| \equiv q \pmod{r}$.

The p-MIN-CMSO problem defined by formula ψ is denoted by p-MIN-CMSO[ψ] and defined as follows.

p-MIN-CMSO[ψ]
Input: A graph G (or digraph D) and an integer k
Parameter: k
Question: Is there a subset $S \subseteq V(G)$ such that $\|S\| \leq k$ and $(G, S) \models \psi$?

In other words, p-MIN-CMSO[ψ] is a subset Π of $\Sigma^* \times \mathbb{Z}$ where for every $(x, k) \in \Sigma^* \times \mathbb{Z}^+$, $(x, k) \in \Pi$ if and only if there exists a set $S \subseteq V$ where $|S| \leq k$ such

that the graph G encoded by x together with S satisfy ψ, i.e., $(G, S) \models \psi$. For $(x, k) \in \Sigma^* \times \mathbb{Z}^-$ we know that $(x, k) \notin \Pi$.

5.6.2 Boundaried Graphs

Here we define the notion of *boundaried graphs* and various operations on them.

Definition 5.6.1. [Boundaried Graphs] *A* boundaried graph *is a graph G with a set $B \subseteq V(G)$ of distinguished vertices and an injective labelling λ from B to the set $\mathbb{Z}^+$. The set B is called the* boundary *of G and the vertices in B are called* boundary vertices *or* terminals*. Given a boundaried graph G, we denote its boundary by $\delta(G)$, we denote its labelling by λ_G, and we define its* label set *by $\Lambda(G) = \{\lambda_G(v) \mid v \in \delta(G)\}$. Given a finite set $I \subseteq \mathbb{Z}^+$, we define $\mathcal{F}_I$ to denote the class of all boundaried graphs whose label set is I. Similarly, we define $\mathcal{F}_{\subseteq I} = \bigcup_{I' \subseteq I} \mathcal{F}_{I'}$. We also denote by $\mathcal{F}$ the class of all boundaried graphs. Finally we say that a boundaried graph is a* t-boundaried *graph if $\Lambda(G) \subseteq \{1, \ldots, t\}$.*

Definition 5.6.2. [Gluing by $\oplus$] *Let G_1 and G_2 be two boundaried graphs. We denote by $G_1 \oplus G_2$ the graph (not boundaried) obtained by taking the disjoint union of G_1 and G_2 and identifying equally-labeled vertices of the boundaries of G_1 and G_2. In $G_1 \oplus G_2$ there is an edge between two labeled vertices if there is either an edge between them in G_1 or in G_2.*

Definition 5.6.3. *Let $G = G_1 \oplus G_2$ where G_1 and G_2 are boundaried graphs. We define the* glued *set of G_i as the set $B_i = \lambda_{G_i}^{-1}(\Lambda(G_1) \cap \Lambda(G_2)), i = 1, 2$. For a vertex $v \in V(G_1)$ we define its* heir *$h(v)$ in G as follows: if $v \notin B_1$ then $h(v) = v$, otherwise $h(v)$ is the result of the identification of v with an equally labeled vertex in G_2. The* heir *of a vertex in G_2 is defined symmetrically. The* common boundary *of G_1 and G_2 in G is equal to $h(B_1) = h(B_2)$ where the evaluation of h on vertex sets is defined in the obvious way. The* heir *of an edge $\{u, v\} \in E(G_i)$ is the edge $\{h(u), h(v)\}$ in G.*

Let $\mathcal{G}$ be a class of (not boundaried) graphs. By slightly abusing notation we say that a boundaried graph *belongs to a graph class* $\mathcal{G}$ if the underlying graph belongs to $\mathcal{G}$.

5.6.3 Finite Integer Index

Definition 5.6.4. [Canonical equivalence on boundaried graphs.] *Let Π be*

a parameterized graph problem whose instances are pairs of the form (G, k). *Given two boundaried graphs* $G_1, G_2 \in \mathcal{F}$, *we say that* $G_1 \equiv_\Pi G_2$ *if* $\Lambda(G_1) = \Lambda(G_2)$ *and there exist a* transposition constant $c \in \mathbb{Z}$ *such that*

$$\forall (F, k) \in \mathcal{F} \times \mathbb{Z} \qquad (G_1 \oplus F, k) \in \Pi \Leftrightarrow (G_2 \oplus F, k + c) \in \Pi.$$

Note that the relation $\equiv_\Pi$ is an equivalence relation. Observe that c could be negative in the above definition. This is the reason we extended the definition of parameterized problems to include negative parameters also.

Next we define a notion of "transposition-minimality" for the members of each equivalence class of $\equiv_\Pi$.

Definition 5.6.5. [Progressive representatives, [24]] *Let* Π *be a parameterized graph problem whose instances are pairs of the form* (G, k) *and let* $\mathcal{C}$ *be some equivalence class of* $\equiv_\Pi$. *We say that* $J \in \mathcal{C}$ *is a* progressive representative *of* $\mathcal{C}$ *if for every* $H \in \mathcal{C}$ *there exists* $c \in \mathbb{Z}^-$, *such that*

$$\forall (F, k) \in \mathcal{F} \times \mathbb{Z} \quad (H \oplus F, k) \in \Pi \Leftrightarrow (J \oplus F, k + c) \in \Pi. \tag{5.1}$$

The following lemma guaranties the existence of a progressive representative for each equivalence class of $\equiv_\Pi$.

Lemma 5.6.1 ([24])**.** *Let* Π *be a parameterized graph problem whose instances are pairs of the form* (G, k). *Then each equivalence class of* $\equiv_\Pi$ *has a progressive representative.*

Definition 5.6.6. [Finite Integer Index, [24, 99]] *For a parameterized problem* Π *and two* t*-boundaried graphs* G_1 *and* G_2, *we say that* $G_1 \equiv_\Pi G_2$ *if there exists a constant* c *such that for every* t*-boundaried graph* G *and for every integer* k, $(G_1 \oplus G_2, k) \in \Pi$ *if and only if* $(G_2 \oplus G, k + c) \in \Pi$. *For every* t, *the relation* $\equiv_\Pi$ *on* t*-boundaried graphs is an equivalence relation, and we call* $\equiv_\Pi$ *the* canonical equivalence relation *of* Π. *We say that a problem* Π *has* Finite Integer Index *if for every* t, $\equiv_\Pi$ *has a finite index on* t*-boundaried graphs.*

Let $\mathcal{G}$ be a graph class. We say that $\mathcal{G}$ is CMSO-*definable* if there exist a sentence ψ on graphs such that $\mathcal{G} = \{G \mid G \models \psi\}$ and, in such a case, we say that ψ defines the class $\mathcal{G}$. Given a parameterized graph problem Π and a graph class $\mathcal{G}$, we denote by $\Pi \Cap \mathcal{G}$ the problem obtained by removing from Π all instances that encode graphs that do not belong to $\mathcal{G}$.

We would like to show that DFVS/DFVS+Tw-η Mod has Finite Integer Index (FII) property over $\mathcal{F}_\eta$. Observe that a digraph $D \in \mathcal{F}_\eta$ if and only if the underlying undirected graph has treewidth at most η. Thus, $\mathcal{F}_\eta$ can be characterized by finite forbidden set of minors. Since minor testing can be expressed into CMSO we have that $\mathcal{F}_\eta$ is CMSO-definable. Let Π denote the DFVS. The problem Π is p-min-CMSO[ψ] and is strongly monotone (see [24] for definition). A proof for this fact is analogous to the proof of [24, Lemma 8.4]. Thus, by [24, Lemmas 7.3 and 7.4] we get the following.

Lemma 5.6.2. *$\Pi \Cap \mathcal{F}_\eta$ has FII. Here, Π is* DFVS/DFVS+Tw-η Mod.

5.6.4 Proof of the Protrusion Replacer Lemma

Proof. The proof is essentially as in [24, Lemma 5.18]. We just adapt it here for our purposes. Our problem is $\Pi \Cap \mathcal{F}_\eta$ where Π is DFVS/DFVS+Tw-η Mod.

We denote by $\mathcal{S}_{\subseteq\{1,\ldots,2t+1\}}$ a set of (progressive) representatives for $\equiv_\Pi$ restricted to boundaried graphs with label sets from $\{1,\ldots,2t+1\}$. Let

$$c = \max\Big\{|V(Y)| \;\Big|\; Y \in \mathcal{S}_{\subseteq\{1,\ldots,2t+1\}}\Big\}.$$

Our algorithm has in its source code hard-wired a table that stores for each boundaried graph G_Y in $\mathcal{F}_{\subseteq\{1,\ldots,2t+1\}}$ on at most $2c$ vertices a boundaried graph $G'_Y \in \mathcal{S}_{\subseteq\{1,\ldots,2t+1\}}$ and a constant $\mu \leq 0$ such that $G_Y \equiv_\Pi G'_Y$, and specifically

(5.2)
$$\forall (F,k) \in \mathcal{F} \times \mathbb{Z} \;:\; (G_Y \oplus F, k) \in (\Pi \Cap \mathcal{F}_\eta) \iff (G'_Y \oplus F, k+\mu) \in (\Pi \Cap \mathcal{F}_\eta).$$

The existence of such a constant $\mu \leq 0$ is guaranteed by the fact that $\mathcal{S}_{\subseteq\{1,\ldots,2t+1\}}$ is a set of progressive representatives.

We now apply [24, Lemma 5.5] and find a $(2t+1)$-protrusion Y of D where $c < |Y| \leq 2c$. Split D into two boundaried graphs $D_Y = D[Y \cup N(Y)]$ and $D_R = D[(V(G)\backslash Y)]$ as follows. Both D_R and D_Y have boundary $N(Y)$, and since $|N(Y)| \leq 2t+1$ we may label the boundaries of D_Y and D_R with labels from $\{1,\ldots,2t+1\}$ such that $D = D_Y \oplus D_R$. As $c < |V(G_Y)| \leq 2c$ the algorithm can look up in its table and find a $D'_Y \in \mathcal{S}_{\subseteq\{1,\ldots,2t+1\}}$ and a constant μ such that $D_Y \equiv D'_Y$ and D_Y, D'_Y and μ satisfy Equation 5.2. Observe that since the initial protrusion X is disjoint from

M we have that the vertex set of M remains in D'. The algorithm outputs

$$(D', k', M) = (D'_Y \oplus D_R, k + \mu, M).$$

Since $|V(D'_Y)| \leq c < |V(D_Y)|$ and $k' \leq k + \mu \leq k$ it remains to argue that the instances (D, k, M) and (D', k', M) are equivalent. However, this is directly implied by Equation 5.2. In particular, by Equation 5.2 we have that D has a directed feedback vertex set of size at most k if and only if D' has directed feedback vertex set of size at most k'. Now we show that $D' - M$ has treewidth at most η. Let $D_R^M = D_R - M$. Now since, $D_Y \equiv D'_Y$ with respect to $\Pi \Cap \mathcal{F}_\eta$ we have that $(D_Y \oplus D_R^M, k^*)$ is in $\Pi \Cap \mathcal{F}_\eta$ if and only if $(D'_Y \oplus D_R^M, k^*)$ is in $\Pi \Cap \mathcal{F}_\eta$. Since $\mathbf{tw}(D_Y \oplus D_R^M) \leq \eta$ we have that $\mathbf{tw}(D'_Y \oplus D_R^M)$ has treewidth at most η. This completes the proof. $\square$

Chapter 6

Kernel for Deletion to Out-Forest

In this chapter (and the next), we study the parameterized complexity of restrictions of DFVS. Recently, in a very interesting article, to make progress on the question of the existence of a polynomial kernel for DFVS, Mnich and van Leeuwen [164] considered DFVS with an additional restriction on *the output rather than the input*. Essentially, the basic philosophy of their program is the following: What happens to the kernelization complexity of DFVS when we consider subclasses of DAGs?

Mnich and van Leeuwen [164] inspected this question by considering the classes of out-forests, out-trees and (directed) pumpkins. An *out-tree* is a digraph where each vertex has in-degree at most 1 and the underlying (undirected) graph is a tree. An *out-forest* is a disjoint union of out-trees. On the other hand, a digraph is a *pumpkin* if it consists of a source vertex s and a sink vertex t, $s \neq t$, together with a collection of internally vertex-disjoint induced directed paths from s to t. Here, all vertices except s and t have in-degree 1 and out-degree 1. The examination of the classes of out-forests and out-trees was also motivated by the corresponding questions of UFVS and Tree Deletion Set in the undirected settings. Formally, Mnich and van Leeuwen [164] studied the following problems.

Out-Forest Vertex Deletion Set (OFVDS) **Parameter:** k
Input: A digraph D and a positive integer k.
Question: Is there a set $S \subseteq V(D)$ of size at most k such that $F = D - S$ is an out-forest?

Out-Tree Vertex Deletion Set (OTVDS) and Pumpkin Vertex Deletion Set (PVDS) are defined in a similar manner, where instead of an out-forest, F

should be an out-tree or a pumpkin, respectively. Mnich and van Leeuwen [164] showed that OFVDS and OTVDS admit kernels of size $\mathcal{O}(k^3)$ and PVDS admits a kernel of size $\mathcal{O}(k^{18})$.

The objective of this chapter is to prove the following results.

- OFVDS admits an $\mathcal{O}(k^2)$ kernel. This result improves upon the best known upper bound of $\mathcal{O}(k^3)$.
- For any $\epsilon > 0$, OFVDS does not admit a kernel of size $\mathcal{O}(k^{2-\epsilon})$ unless $\mathsf{coNP} \subseteq \mathsf{NP/poly}$.

In the next chapter, we give a $\mathcal{O}(k^3)$ kernel for PVDS.

To get the improved kernel for OFVDS we incorporate the Expansion Lemma as well as a factor 3-approximation algorithm for OFVDS in the kernelization routine given in [164]. The significance of this improvement also lies in the fact that it is essentially tight.

6.1 Improved Kernel for Out-Forest Vertex Deletion Set

The aim of this section is to present an $\mathcal{O}(k^2)$ kernel for OFVDS. In Section 6.1.1 we state definitions and results relevant to our kernelization algorithm. Next, in Section 6.1.2, we design an algorithm for OFVDS that outputs a 3-approximate solution, which will also be used by our kernelization algorithm. Finally, in Section 6.1.3, we present our kernelization algorithm.

6.1.1 Some Prerequisites

A *collision* is a triplet (u, w, v) of distinct vertices such that $(u, w), (v, w) \in E(D)$. We start by giving the definition of a q-expansion and the statement of the Expansion Lemma.

Definition 6.1.1 (q-Expansion)**.** *For a positive integer q, a set of edges $M \subseteq E(G)$ is a* q-expansion of A into B *if (i) every vertex in A is incident to exactly q edges in M, and (ii) M saturates exactly $q|A|$ vertices in B (i.e., there is a set of $q|A|$ vertices in B which are incident to edges in M).*

Lemma 6.1.1 (Expansion Lemma [65, 201])**.** *Let q be a positive integer and G be an undirected bipartite graph with vertex bipartition (A, B) such that $|B| \geq q|A|$, and*

there are no isolated vertices in B. Then, there exist non-empty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that there exists a q-expansion of X into Y, and no vertex in Y has a neighbor outside X (i.e., $N(Y) \subseteq X$). Furthermore, the sets X and Y can be found in time polynomial in the size of G.

We will also need to rely on the well-known notion of l-flowers.

Definition 6.1.2 (l-Flower)**.** *An undirected graph G contains an* l-flower through v *if there is a family of cycles $\{C_1, \ldots, C_l\}$ in G such that for all distinct $i, j \in [l]$, $V(C_i) \cap V(C_j) = \{v\}$.*

Lemma 6.1.2 ([65, 201])**.** *Given an undirected graph G and a vertex $v \in V(G)$, there is a polynomial-time algorithm that either outputs a $(k+1)$-flower through v or, if no such flower exists, outputs a set $Z_v \subseteq V(G) \setminus \{v\}$ of size at most $2k$ that intersects every cycle that passes through v in G.*

6.1.2 Approximation Algorithm for Out-Forest Vertex Deletion Set

This section presents a 3-factor approximation algorithm for OFVDS. Given an instance of OFVDS, let OPT be the minimum size of a solution. Formally, we solve the following.

3-Approximate Out-Forest Vertex Deletion Set (Approx-OFVDS)
Input: A directed graph D.
Output: A subset $X \subseteq V(D)$ such that $D-X$ is an out-forest and $\lvert X\rvert \leq 3 \cdot OPT$.

Given three distinct vertices $u_1, u_2, u_3 \in V(D)$, we say (u_1, u_2, u_3) is a *collision* if u_1 and u_2 are in-neighbors of u_3. Observe that any solution to OFVDS (and hence, Approx-OFVDS) must intersect any collision in at least 1 vertex. Moreover, it must intersect any cycle in at least 1 vertex. These observations form the basis of this algorithm.

Lemma 6.1.3. Approx-OFVDS *can be solved in polynomial time.*

Proof. Given a digraph D, the algorithm first constructs (in polynomial time) a family $\mathcal{F}$ of collisions and induced cycles in D such that the vertex sets of the entities in this family are pairwise disjoint. To do this, it initializes $\mathcal{F} = \emptyset$. Then, as

long as there exists a vertex $v \in V(D)$ with at least two in-neighbors, u_1 and u_2, it inserts (v, u_1, u_2) into $\mathcal{F}$ and removes v, u_1 and u_2 from D (only for the purpose of the construction of $\mathcal{F}$). Once there is no vertex $v \in V(D)$ such that $d^-(v) \geq 2$, observe that all the directed cycles in the resulting digraph are pairwise vertex disjoint. $\mathcal{F}$ additionally contains all these pairwise vertex disjoint directed cycles..

Let us now construct a solution, S_{app}, for Approx-OFVDS. For every collision in $\mathcal{F}$, we let S_{app} contain each of the three vertices of this collision. From every cycle C in $\mathcal{F}$ we pick an arbitrary vertex and insert it into S_{app}. Clearly, $|S_{app}| \leq 3|\mathcal{F}|$. Since $\mathcal{F}$ is a collection of pairwise vertex disjoint collisions and directed cycles, $|\mathcal{F}| \leq OPT$. Thus, $|S_{app}| \leq 3 \cdot OPT$. It is now sufficient to prove that $D - S_{app}$ is an out-forest. Observe that no vertex v in $D - S_{app}$ has in-degree at least 2, otherwise the collision consisting of v and two of its in-neighbors would have been inserted into $\mathcal{F}$ and hence also into S_{app}. Moreover, there is no directed cycle C in $D - S_{app}$. Indeed, if the cycle C intersects an collision in $\mathcal{F}$, it is clear that it cannot exist in $D - S_{app}$, and otherwise it would have been inserted into $\mathcal{F}$ and hence one of its vertices would have been inserted into S_{app}. We thus conclude that the theorem is correct. □

6.1.3 Kernelization Algorithm for Out-Forest Vertex Deletion Set

We are now ready to present our kernelization algorithm. Let (D, k) be an instance of OFVDS. We note that during the execution of our algorithm, D may become a multigraph.

Preprocessing. We start by applying the following reduction rules exhaustively, where a rule is applied only if its condition is true and the conditions of all of the preceding rules are false. Rule 6.1.4 is given in [164], and its correctness is proven in that paper. It will be clear that the first five rules can be applied in polynomial time, whereas, for applying the last rule, we call the algorithm given by Lemma 6.1.2. Moreover, it is straightforward to verify that each of these rules, except Rule 6.1.4 (whose safeness follows from [164]), is safe (i.e., the instance it returns is equivalent to the input instance).

Reduction Rule 6.1.1. *If there exists a vertex $v \in V(D)$ such that $d^+(v) = 0$ and $d^-(v) \leq 1$, remove v from D.*

Reduction Rule 6.1.2. *If there exists a directed path $P = (w_0, w_1, \ldots, w_l, w_{l+1})$ in*

D such that $l \geq 2$ and for all $i \in [l]$, $d^-(w_i) = d^+(w_i) = 1$, remove each vertex in $\{w_1, \ldots, w_{l-1}\}$ from D and add the edge (w_0, w_l) to D.

Reduction Rule 6.1.3. *If there exists an edge $(u, v) \in E(D)$ with multiplicity at least 3, remove all but two copies of it.*

Reduction Rule 6.1.4. *If there exist $k+1$ collisions $(u_1, w_1, v), \ldots, (u_{k+1}, w_{k+1}, v)$ that pairwise intersect only at v, remove v from D and decrease k by 1.*

Reduction Rule 6.1.5. *If there exists a vertex $v \in V(D)$ such that $d^-(v) \geq k+2$, remove v from D and decrease k by 1.*

Reduction Rule 6.1.6. *Let G be the underlying graph of D. If there exists a vertex $v \in V(G)$ such that there is a $(k+1)$-flower through v in G, remove v from D and decrease k by 1.*

Bounding Out-Degrees. Next, we aim to bound the maximum out-degree of a vertex in D. To this end, suppose that there exists a vertex $v \in V(D)$ with $d^+(v) \geq 16k$. Let G be the underlying graph of D. Since Reduction Rule 6.1.6 is not applicable, we let Z_v be the set obtained by calling the algorithm given by Lemma 6.1.2. Moreover, we let S_{app} be a 3-factor approximate solution obtained by calling the algorithm given by Theorem 6.1.3. We can assume that $|S_{app}| \leq 3k$, since otherwise the input instance is a NO instance. Denote $X_v = (S_{app} \cup Z_v) \setminus \{v\}$. Since $|Z_v| \leq 2k$, we have that $|X_v| \leq 5k$.

We proceed by examining the set $\mathcal{C}_v = \{C_1, C_2, \ldots, C_{|\mathcal{C}_v|}\}$ of the connected components in $G - (X_v \cup \{v\})$. Since S_{app} is an approximate solution, each component $C_i \in \mathcal{C}_v$ is an out-tree. Moreover, for any component $C_i \in \mathcal{C}_v$, v has at most one neighbor in C_i, since otherwise there would have been cycle passing through v in $G - Z_v$, contradicting the definition of Z_v. For each component $C_i \in \mathcal{C}_v$, let u_i be the root of C_i. Let $\mathcal{D}_v = \{C_i \mid C_i \in \mathcal{C}_v, (v, u_i) \in E(D)\}$ and $\tilde{\mathcal{D}}_v = \{C_i \mid C_i \in \mathcal{C}, (v, u) \in E(D), u \in V(C_i), u \neq u_i\}$. Observe that $d^+(v) \leq |\mathcal{D}_v| + |\tilde{\mathcal{D}}_v| + |X_v|$. Moreover, since Reduction Rule 6.1.4 is not applicable, $|\tilde{\mathcal{D}}_v| \leq k$. Since $d^+(v) \geq 16k$, we have that $|\mathcal{D}_v| \geq 10k$. Without loss of generality, let $\mathcal{D}_v = \{C_1, \ldots, C_p\}$ where $p = |\mathcal{D}_v|$. Since Reduction Rule 8.3.3 is not applicable, for any component $C_i \in \mathcal{D}_v$ there exists an edge in $E(G)$ with one endpoint in C_i and the other in X_v.

We now construct an auxiliary (undirected) bipartite graph H with bipartition (A, B), where $A = X_v$ and B is a set of new vertices denoted by $b_1, \ldots, b_p$. For any $u \in A$ and $b_i \in B$, $(u, b_i) \in E(H)$ if and only if there exists an edge in G between

u and some vertex in C_i. Since $|B| \geq 2|A|$ and there are no isolated vertices in B, we can use the algorithm given by Lemma 6.1.1 to obtain non-empty vertex sets $X'_v \subseteq A$ and $Y'_v \subseteq B$ such that there is a 2-expansion of X'_v into Y'_v and $N(Y'_v) \subseteq X'_v$. Let $\mathcal{D}'_v = \{C_i \mid b_i \in Y'_v\}$.

Reduction Rule 6.1.7. *Remove each of the edges in D between v and any vertex in a component in $\mathcal{D}'_v$. For every vertex $x_i \in X'_v$, insert two copies of the edge (v, x_i) into $E(D)$.*

Lemma 6.1.4. *Reduction Rule 6.1.7 is safe.*

Proof. Let D' be the graph resulting from the application of the rule. We need to prove that (D, k) is a YES instance if and only if (D', k) is a YES instance.

Forward Direction. For the forward direction, we first claim that if (D, k) has a solution S such that $v \notin S$, then it has a solution S' such that $X'_v \subseteq S'$. To this end, suppose that (D, k) has a solution S such that $v \notin S$. Let $S' = (S \setminus \bigcup_{C_i \in \mathcal{D}'_v} V(C_i)) \cup X'_v$. It holds that $|S'| \leq |S|$ since for each $x \in X'_v \setminus S$, at least one vertex from at least one of the two components in its expansion set must belong to the solution. Suppose for the sake of contradiction that $F = D - S'$ is not an out-forest. First, assume that there exists a vertex in F with in-degree at least 2. Note that $V(D) = \bigcup_{C_i \in \mathcal{C}_v} V(C_i) \cup X_v \cup \{v\}$. Recall that the neighborhood of each of the vertices in the connected components that belong to $\mathcal{D}'_v$, outside their own component, is contained in $\{v\} \cup X'_v$. Moreover, v only has out-neighbors in the components that belong to $\mathcal{D}'_v$ and each $C_i \in \mathcal{C}_v$ is an out-tree. Therefore, since $D - S$ has no vertex of in-degree at least 2, so does $D - S'$. Now, assume that there is a cycle C in F. Then, if $V(C) \cap (S \cap \bigcup_{C_i \in \mathcal{D}'} V(C_i)) = \emptyset$, then C is also a cycle in $D - S$, which is a contradiction. Thus, $V(C) \cap (S \cap \bigcup_{C_i \in \mathcal{D}'} V(C)_i) \neq \emptyset$. However, any cycle that passes through a component in $\mathcal{D}'_v$ also passes through v and a vertex in X'_v. Since $X'_v \subseteq S'$, no such cycle exists. This finishes the proof of the claim.

Let S be a solution to (D, k). If $v \in S$, then it is clear that $D' - S$ is an out-forest. Otherwise, if $v \notin S$, our claim implies that (D, k) has a solution S' such that $X'_v \subseteq S'$. Then, $D' - S'$ is an out-forest.

Backward Direction. For the backward direction, let us prove the following claim. If (D', k) has a solution S such that $v \notin S$, then $X'_v \subseteq S$. Suppose, by way of contradiction, that the claim is incorrect. Then, there exists $x \in X'_v$ such that $x \notin S$. However, this implies that $D' - S$ is not an out-forest as it contains the double edges (v, x_i).

Now, let S be a solution to (D', k), and denote $F = D' - S$. Suppose $v \in S$. Then observe that, $D - S = D' - S$ is an out forest and thus S is solution to (D, k). If $v \notin S$, then by our previous claim, $X'_v \subseteq S$. Recall that for each $C_i \in \mathcal{C}_v$, C_i is an out-tree and u_i is the root of the out-tree C_i. Observe that for each $C_i \in \mathcal{D}'_v$ such that $u_i \notin S$, u_i is a root of some out-tree in F. This, is because the neighborhood of such a u_i in $D - C_i$, is contained in $X'_v \cup \{v\}$, $X'_v \subseteq S$ and each edge between v and any vertex in $\mathcal{D}'_v$ is absent in D'. Also observe that, each such vertex u_i and v belong to different out-trees of F because in D', the edges between v and any vertex in $\mathcal{D}'_v$ are absent. This implies that if we add (to D') the edges between v and each vertex u_i that have been removed by the application of the rule, $V(F)$ will still induce an out-forest. Thus, S is a solution to (D, k). □

After an exhaustive application of Reduction Rule 6.1.7, the out-degree of each vertex in D is at most $16k - 1$. However, since this rule inserts edges into $E(G)$, we need the following lemma.

Lemma 6.1.5. *The total number of applications of the reduction rules is bounded by a polynomial in the input size.*

Proof. We associate the measure $\mu(D, k) = |V(D)| + |E^1(D)|$ with every instance, where $E^1(D)$ are those edges in $E(D)$ that have multiplicity 1. Initially, this measure is bounded by a polynomial in the input size. We maintain the invariant that the measure is always positive and it strictly decreases whenever any of the reduction rules is applied. We thus conclude that the lemma is correct. □

Correctness. By relying on counting arguments as well as Lemmas 6.1.4 and 6.1.5, we obtain the main result of this section.

Theorem 6.1.1. OFVDS *admits an* $\mathcal{O}(k^2)$*-kernel.*

Proof. Since our reduction rules are safe and can be applied only polynomially many times, we next assume that none of them is applicable, and turn to bound the number of vertices and edges in the resulting instance (D, k). To this end, suppose that (D, k) is a YES instance. Let S be a solution, and denote $F = D - S$. Let $V_{\leq 1}$ be the vertices in F that have degree at most 1 (in F), $V_{=2}$ be the vertices in F that have degree exactly 2, and $V_{\geq 3}$ be the vertices in F that have degree at least 3. Note that $|V(F)| = |V_{\leq 1}| + |V_{=2}| + |V_{\geq 3}|$.

Since Reduction Rules 6.1.5 and 6.1.7 are not applicable, the total degree of any vertex is bounded by $\mathcal{O}(k)$. In particular, the number of neighbors in F of each

vertex in S is bounded by $\mathcal{O}(k)$. Therefore, the total number of vertices in F that have a neighbor in S is bounded by $\mathcal{O}(k^2)$. Observe that each vertex in $V_{\leq 1}$ is either a leaf in F or a root in F. Also all leaf vertices of F are in $V_{\leq 1}$. If a vertex in $V_{\leq 1}$ is a leaf in F, then such a vertex must have at least one neighbor in S, as otherwise Reduction Rule 8.3.3 would have been applicable. Thus, the total number of leaf vertices in $|V_{\leq 1}|$ is bounded by $\mathcal{O}(k^2)$. Since the total number of vertices that are a root of some out-tree in F are at most the number of leaves in F, $V_{\leq 1}$ is bounded by $\mathcal{O}(k^2)$. Since the underlying graph of F is a forest, $|V_{\geq 3}| < |V_{\leq 1}|$. Thus, $V_{\geq 3}$ is also bounded by $\mathcal{O}(k^2)$. Let us mark each of the vertices in F which has a neighbor in S as well as the vertices in $V_{\geq 3}$. Note that all other vertices are degree-2 vertices in D. Let $\mathcal{P}$ be the set maximal degree-2 paths in F whose endpoints are not marked. Observe that $|\mathcal{P}|$ is bounded by $\mathcal{O}(k^2)$. Moreover, the length of a maximal degree-2 path in $\mathcal{P}$ is at most 1, else Reduction Rule 8.3.4 is applicable.

We deduce that the number of vertices in D should be bounded by $\mathcal{O}(k^2)$, otherwise we can conclude that (D, k) be a No instance. Since F is a forest, $|E(F)| < |V(F)|$. Moreover, the number of edges incident to vertices in S is bounded by $\mathcal{O}(k^2)$. Therefore, the number of edges in D should be bounded by $\mathcal{O}(k^2)$. □

In the next section, we prove that the size of the kernel given in Theorem 6.1.1 is tight, that is OFVDS does not admit an $\mathcal{O}(k^{2-\epsilon})$ size kernel unless $\mathsf{coNP} \subseteq \mathsf{NP/poly}$.

6.2 Kernel Lower Bound for Out-Forest Vertex Deletion Set

In this section we show that the size of our kernel for OFVDS is optimal i.e., there is no kernel for OFVDS which can be encoded into $\mathcal{O}(k^{2-\epsilon})$ bits for any $\epsilon > 0$ unless $\mathsf{co\text{-}NP} \subseteq \mathsf{NP/poly}$. To this end, we will rely on the fact that Vertex Cover (VC) does not admit a kernel which can be encoded into $\mathcal{O}(k^{2-\epsilon})$ bits for any $\epsilon > 0$ unless $\mathsf{co\text{-}NP} \subseteq \mathsf{NP/poly}$ [74].[1]

We give a polynomial-time reduction from VC to OFVDS. Let (G, k) be an instance of VC. We construct an instance of (D, k') of OFVDS as follows. Initially, $V(D) = V(G)$ and $E(D) = \emptyset$. For each edge $\{u, v\} \in E(G)$, we add a vertex e_{uv} to $V(D)$, and we add the edges (u, e_{uv}) and (v, e_{uv}) to $E(D)$. Finally, we set $k' = k$.

[1]Given a graph G and a parameter k, VC asks whether one can remove at most k vertices from G so that the resulting graph will be edge-less.

Lemma 6.2.1. (G, k) *is a* YES *instance of* VC *if and only if* (D, k') *is a* YES *instance of* OFVDS.

Proof. In the forward direction, consider a vertex cover S of G of size at most k. We claim that $D - S$ is an out-forest. By construction, each vertex $v \in V(D)$ that also belongs to G has in-degree 0 in D. For each vertex $e_{uv} \in V(D)$, $\{u, v\} \in E(G)$, and therefore at least one vertex among u and v must belong to S. Moreover, u and v are the only in-neighbors of e_{uv}. Therefore, for all $e_{uv} \in V(D)$, the in-degree of e_{uv} is at most 1 in $D - S$. Note that $V(G) \subseteq V(D)$ is an independent set in D. Similarly, $V(D) \setminus V(G)$ is an independent set in D. Moreover, each vertex in $V(D) \setminus V(G)$ has out-degree 0. Therefore, $D - X$ is acyclic.

In the reverse direction, consider an out-forest deletion set X of D of size at most k. Observe that if $e_{uv} \in X$, then $(X \setminus \{e_{uv}\}) \cup \{u\}$ is also an out-forest deletion set in D. Hence, without loss of generality assume that $X \subseteq V(G)$. Suppose, by way of contradiction, that X is not a vertex cover of G. Then, there is an edge $(u, v) \in E(G)$ such that $X \cap \{u, v\} = \emptyset$. By assumption X does not contain e_{uv}. Therefore, e_{uv} is a vertex with in-degree 2 in $D - X$, which contradicts the choice of X. □

We will use the Theorem 6.2.1 along with Lemma 6.2.1 to rule out an $\mathcal{O}(k^{2-\epsilon})$ kernel for OFVDS. For this purpose, we first state the definition of Oracle Communication Protocol [74].

Definition 6.2.1. *An oracle communication protocol for a language L is a communication protocol between two players. The first player is given the input x and has to run in time polynomial in the length of the input; the second player is computationally unbounded but is not given any part of x. At the end of the protocol the first player should be able to decide whether $x \in L$. The cost of the protocol is the number of bits of communication from the first player to the second player.*

Theorem 6.2.1 (Theorem 2 [74])**.** *Let $d \geq 2$ be an integer and ϵ a positive real number. If* coNP $\not\subseteq$ NP/poly*, there is no protocol of cost $\mathcal{O}(n^{d-\epsilon})$ to decide whether a d-uniform hypergraph on n vertices has a vertex cover of at most k vertices, even when the first player is co-nondeterministic.*

Theorem 6.2.2. *Let $\epsilon > 0$.* OFVDS *does not admit an oracle communication protocol of cost $\mathcal{O}(k^{2-\epsilon})$ for deciding (D, k) unless* coNP $\subseteq$ NP/poly.

Proof. Suppose OFVDS admits an oracle communication protocol of cost $\mathcal{O}(k^{2-\epsilon})$ for deciding (D, k). Combining this supposition with Lemma 6.2.1, we get a protocol of cost $\mathcal{O}(k^{2-\epsilon})$ for VC. By Theorem 6.2.1, this implies that $\mathsf{coNP} \subseteq \mathsf{NP/poly}$. □

We are now ready to prove the main result of this section.

Theorem 6.2.3. OFVDS *does not admit a kernel of size* $\mathcal{O}(k^{2-\epsilon})$ *for any* $\epsilon > 0$ *unless* $\mathsf{coNP} \subseteq \mathsf{NP/poly}$.

Proof. If OFVDS admits a kernel of size $\mathcal{O}(k^{2-\epsilon})$, then the computationally bounded player computes the kernel and sends it to the computationally unbounded player who can correctly compute and return one bit answer. The cost of this protocol is $\mathcal{O}(k^{2-\epsilon})$. Therefore, by Theorem 6.2.2 it implies that $\mathsf{coNP} \subseteq \mathsf{NP/poly}$. □

Chapter 7

Kernel for Deletion to Pumpkin

Recall from the last chapter that a digraph is a *pumpkin* if it consists of a source vertex s and a sink vertex t, $s \neq t$, together with a collection of internally vertex-disjoint induced directed paths from s to t. Here, all vertices except s and t have in-degree 1 and out-degree 1. The Pumpkin Vertex Deletion Set problem is defined again below.

Pumpkin Vertex Deletion Set (PVDS) **Parameter:** k
Input: A digraph D and a positive integer k.
Question: Is there a set $S \subseteq V(D)$ of size at most k such that $F = D - S$ is an out-forest?

Theorem 7.0.1. PVDS *admits an* $\mathcal{O}(k^3)$*-vertex kernel.*

Recall the motivations of designing an improved kernel for PVDS from the last chapter. In this chapter, we prove the following theorem.

7.1 Outline of the Kernelization Algorithm for Pumpkin Vertex Deletion Set

The kernelization algorithm for PVDS given in [164] works roughly as follows. It has two phases: (a) first it gives an $\mathcal{O}(k^5)$ kernel for a variant of the problem where we know the source and the sink of the pumpkin obtained after deleting the solution vertices; and (b) in the second phase, it reduces PVDS to polynomially many instances of a variant of the problem mentioned in item (a) and then composes these instances to get a kernel of size $\mathcal{O}(k^{18})$. In fact given an instance (D, k) of PVDS, the kernelization algorithm of [164] outputs an equivalent instance (D', k') such that $k' = \mathcal{O}(k^{18})$. We take a completely different route and use "sun-flower

style" reduction rules together with a marking strategy to obtain an equivalent instance (D', k') such that $|V(D')| + |E(D')| = \mathcal{O}(k^3)$ and $k' \leq k$. We believe the method applied in this algorithm could be useful also in other kernelization algorithms.

Let (D, k) be an instance of PVDS. We assume that $|V(D)| \geq k^3$, else we are done. Let $\mathsf{HO} = \{v \in V \mid d^+(v) \geq k+2\}$ and $\mathsf{HI} = \{v \in V \mid d^-(v) \geq k+2\}$. That is, HO and HI are vertices of high out-degrees and high in-degrees, respectively. Mnich and Leeuwen [164] proved that the following reduction rule is safe.

Reduction Rule 7.1.1. *If $|\mathsf{HO}| > k+1$ or $|\mathsf{HI}| > k+1$, return that (D, k) is a* No *instance.*

For the sake of clarity, we divide the presentation of the kernelization algorithm into two subsections. At the end of Section 7.2, we will simplify the instance in a way that will allow us to assume that if there is a solution S, then *both the source and sink of the pumpkin $D - S$ belong to* $\mathsf{HO} \cup \mathsf{HI}$ (Assumption 7.2.1). This assumption will be at the heart of the "marking approach" of Section 7.3, which will handle instances which have been reduced with respect to the reduction rules in Section 7.2. An intuitive explanation of the necessity of our marking process is given at the beginning of Section 7.3. Throughout this section, if k becomes negative, we return that (D, k) is a No instance, and if D becomes a pumpkin and k is positive or zero, we return that (D, k) is a Yes

instance.

7.2 The Simplification Phase

For any $v \in V(D)$, denote by X_v the set of in-neighbors of v, that is, $X_v = N^-(v)$ and by Y_v the set of every vertex $y \in V(D) \setminus \{v\}$ for which there exists a vertex $x \in X_v$ such that $(x, y) \in E(D)$. Note that X_v and Y_v may or may not be disjoint sets. We now give a construction of an auxiliary graph that will be used to prove the safeness of the upcoming reduction rule. For this, consider a set Y'_v of new vertices such that there is exactly one vertex $y' \in Y'_v$ for any $y \in Y_v$. That is, Y'_v is a set containing a copy for each of the vertex in Y_v. By construction, X_v and Y'_v are disjoint sets. Let H_v^- be the (undirected) bipartite graph on the vertex set $X_v \cup Y'_v$ where for all $x \in X_v$ and $y' \in Y'_v$, $\{x, y'\} \in E(H_v^-)$ if and only if $(x, y) \in E(D)$ (see Fig. 7.1). Let $\mathsf{match}^-(v)$ be the size of a maximum matching in H_v^-.

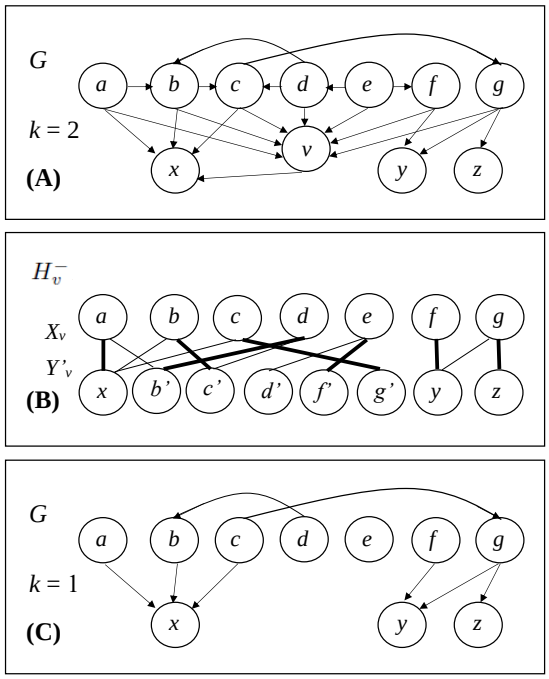


Figure 7.1: (A) An instance of PVDS. (B) The graph H_v^-. The bold edges correspond to a maximum matching. (C) An application of Rule 7.2.1.

An application of the next rule is illustrated in Fig. 7.1.

Reduction Rule 7.2.1. *If there exists a vertex $v \in V(D)$ such that* $\mathsf{match}^-(v) \geq 2(k+1)$*, remove v from D and decrease k by* 1.

Lemma 7.2.1. *Reduction Rule 7.2.1 is safe.*

Proof. For the backward direction, trivially if S is a pumpkin deletion set in $D-\{v\}$ of size at most $k-1$, then $S \cup \{v\}$ is a pumpkin deletion set in D of size at most k. For the forward direction, it is *sufficient* to show that if (D,k) is a YES instance then *every solution S contains v*. For a contradiction, assume that there exists a solution S that does not contain v. Let M be a maximum matching in the graph H_v^-. Observe that for every edge $\{x,y'\} \in M$ where $x \in X_v$, if x is not the source of the pumpkin $D-S$, it holds that $|S \cap \{x,y\}| \geq 1$ (otherwise the pumpkin $D-S$ contains a vertex, which is not its source, and has at least two out-neighbors). Moreover, for every edge $\{x,y'\} \in M$ where $x \in X_v$, if y is the source of the pumpkin $D-S$, it holds that $x \in S$. We thus deduce that *for all but one of the edges $\{x,y'\} \in M$, we have that* $|S \cap \{x,y\}| \geq 1$. Since M is a matching, for every vertex $u \in S$, the vertex u can belong to at most one edge in M, and the vertex u' (if it belongs to Y'_v) can also belong to at most one edge in M. However, $|S| \leq k$,

and therefore $S \cup \{y' \in Y'_v : y \in S\}$ can intersect at most $2k$ edges in M. Since $S \cup \{y' \in Y'_v : y \in S\}$ must intersect all but one edge of M and $|M| \geq 2(k+1)$, we obtain a contradiction. □

Now, to present the symmetric rule, for any vertex $v \in V(D)$, denote by X_v the set of out-neighbors of v, that is, $X_v = N^+(v)$. Let Y_v be the set of vertices $y \in V(D)$ for which there exists a vertex $x \in X_v$ such that $(y, x) \in E(D)$. Let Y'_v be a set containing a copy y' of each vertex $y \in Y$. Let H^+_v be the bipartite graph on the vertex-set $X_v \cup Y'_v$ which for all $x \in X_v$ and $y' \in Y'_v$ contains the edge $\{x, y'\}$ if and only if $(y, x) \in E(D)$. Let $\mathsf{match}^+(v)$ be the size of a maximum matching in H^+_v. Then, the following reduction rule is safe.

Reduction Rule 7.2.2. *If there exists a vertex $v \in V(D)$ such that $\mathsf{match}^+(v) \geq 2(k+1)$, remove v from D and decrement k by 1.*

We also need the following rule, proved by Mnich and Leeuwen [164].

Reduction Rule 7.2.3. *Let $P = (w_0, \cdots, w_\ell)$ be an induced directed path, that is for all $i \in [l-1]$ $d^-(w_i) = d^+(w_i) = 1$, with $\ell > k+2$ in D. Then, delete w_1 from D and add the edge (w_0, w_2).*

Consider some hypothetical solution S (if such a solution exists). Let s and t denote the source and sink, respectively, of the pumpkin $D - S$. Let A (or B) denote the set of out-neighbors (resp. in-neighbors) of s (resp. t) in the pumpkin. Clearly, $|A| = |B|$. Let $C = V(D) \setminus (S \cup A \cup B \cup \{s, t\})$. Next, we prove a series of useful claims relating to S.

Lemma 7.2.2. *(i) Every vertex in $\{s\} \cup A \cup B \cup C$ has in-degree (in D) at most $k+1$, and (ii) every vertex in $\{t\} \cup A \cup B \cup C$ has out-degree (in D) at most $k+1$.*

Proof. The correctness of the claim follows from the observation that every vertex in $\{s\} \cup A \cup B \cup C$ has at most one in-neighbor in the pumpkin $D - S$ and at most $|S| \leq k$ in-neighbors outside this pumpkin, and every vertex in $\{t\} \cup A \cup B \cup C$ has at most one out-neighbor in this pumpkin and at most $|S| \leq k$ in-neighbors outside this pumpkin. □

Lemma 7.2.3. *For any vertex $v \in V(D)$, $|N^-(v) \cap C|, |N^+(v) \cap C| \leq 2(k+1)$.*

Proof. We only show that $|N^-(v) \cap C| \leq 2(k+1)$, since the proof of the other inequality is symmetric. Clearly, any vertex $v \in \{s, t\} \cup A$ has no in-neighbor

among the vertices in C and any vertex $v \in C \cup B$ has at most one in-neighbor among the vertices in C. Thus, we are left with showing the claim only for vertices belonging to S. Let $v \in S$ and for a contradiction assume that $|N^-(v) \cap C| > 2k+2$. We will show that in this case Rule 7.2.1 is applicable. Let X_v^C denote the set of in-neighbors of v in C. That is, $X_v^C = N^-(v) \cap C$. Consider the graph H_v^-. Now observe that $X_v^C \subseteq X_v$ and all the out-neighbors of vertices in X_v^C that belong to $D - S$ are in $C \cup B$ (recall that S is the hypothetical solution we are working with). Observe that each vertex in C has a unique out-neighbor in $D - S$ and in particular they belong to $C \cup B$. Thus, all the out-neighbors of vertices in X_v^C belonging to $C \cup B$, say Y_v^C, belong to Y_v. Recall that H_v^- is a bipartite graph with the vertex bipartition X_v and Y_v' (where in Y_v' we make a copy for each of the vertices in Y_v), and thus we have that $|\mathsf{match}^-(v)| \geq |X_v^C| > 2(k+1)$. Therefore, it is not possible that $|X_v^C| = |N^-(v) \cap C| > 2k+2$, since in this case Rule 7.2.1 is applicable. This completes the proof. □

The set of in-neighbors (or out-neighbors) of any vertex $v \in V(D)$ is contained in $A \cup B \cup C \cup S \cup \{s, t\}$. Since $|A| \leq d^+(s)$, $|B| \leq d^-(t)$ and $|S| \leq k$, Lemma 7.2.3 gives us the following corollary.

Corollary 7.2.1. *For any vertex $v \in V(D)$, $d^-(v), d^+(v) \leq 3k + d^+(s) + d^-(t) + 4$.*

We further strengthen this corollary to obtain the following result.

Lemma 7.2.4. *For any vertex $v \in V(D)$, $d^-(v), d^+(v) \leq \min\{4k + 2d^+(s) + 4, 4k + 2d^-(t) + 4\}$.*

Proof. Let $v \in V(D)$. By Corollary 7.2.1, $d^-(v), d^+(v) \leq 3k + d^+(s) + d^-(t) + 4$. Thus, to prove the claim, it is sufficient to show that (i) $d^-(t) \leq d^+(s) + k$, and (ii) $d^+(s) \leq d^-(t) + k$. We only consider the first item, since the proof of the second one is symmetric. Observe that the pumpkin $D - S$ contains at most $d^+(s)$ internally vertex disjoint paths from s and t. Therefore, since at most k in-neighbors of t can belong to S, t has (in D) at most $d^+(s) + k$ in-neighbors. □

Let $M = \max_{v \in V(D)}\{d^+(v), d^-(v)\}$. The next corollary (Corollary 7.2.2), derived from Lemma 7.2.4, and rule (Reduction Rule 7.2.4) will bring us to the main goal of this subsection, summarized in Assumption 7.2.1 below.

Corollary 7.2.2. *If $M > 6k + 6$, then $s \in \mathsf{HO}$ and $t \in \mathsf{HI}$.*

Proof. Suppose that $M > 6k+6$. Let $v \in V(D)$ be a vertex such that $6k+6 < M = \max\{d^+(v), d^-(v)\}$. By Lemma 7.2.4, $d^-(v), d^+(v) \leq \min\{4k+2d^+(s)+4, 4k+2d^-(t)+4\}$, and therefore $6k+6 < \min\{4k+2d^+(s)+4, 4k+2d^-(t)+4\}$. We thus get that $d^+(s), d^-(t) > k+1$, which implies that $s \in \mathsf{HO}$ and $t \in \mathsf{HI}$. □

Reduction Rule 7.2.4. *If $|V(D)| > 2k^2M + 4kM + k + 2$, return (D, k) is a* No *instance.*

Lemma 7.2.5. *Reduction Rule 7.2.4 is safe.*

Proof. Suppose that the instance is a Yes instance. Let S be a solution. Then, the pumpkin $D - S$ contains at most $2M|S| \leq 2Mk$ vertices with at least one neighbor in S. Let Z be the set of these vertices. Then, $V(D) \setminus (S \cup Z \cup \{s,t\})$ is a collection of at most $(2k+1)M$ paths whose vertices (including the endpoints) are vertices of in-degree 1 and out-degree 1 in D. By Rule 7.2.3, each such path contains at most k vertices. Therefore, $|V(D)| \leq (2k+1)kM + |Z| + |S \cup \{s,t\}| \leq (2k+1)kM + 2kM + k + 2 = 2k^2M + 4kM + k + 2$. □

By Rule 7.2.4, if $M \leq 6k+6$, we obtain the desired kernel. Thus, by Corollary 7.2.2, we have the following observation.

Assumption 7.2.1. *From now on, we can assume that if a solution exists, in the resulting pumpkin the source belongs to HO and the target belongs to HI.*

Next, it will be convenient to assume that HI and HO are disjoint sets. To this end, we apply the following rule exhaustively, where safeness follows directly from Lemma 7.2.2.

Reduction Rule 7.2.5. *Remove all vertices in $HI \cap HO$ and decrease k by $|\mathsf{HI} \cap \mathsf{HO}|$.*

We will also assume that the following rule, illustrated in Fig. 7.2, has been applied exhaustively. This assumption will be used at the end of the following subsection (in the proof of Lemma 7.3.7).

Reduction Rule 7.2.6. *If there exists a vertex $v \notin \mathsf{HI} \cup \mathsf{HO}$ such that $N^-(v) \cap (V(D) \setminus \mathsf{HI}) = \emptyset$ or $N^+(v) \cap (V(D) \setminus \mathsf{HO}) = \emptyset$, delete v from D and decrease k by 1.*

Lemma 7.2.6. *Reduction Rule 7.2.6 is safe.*

Proof. We only consider the case where there exists a vertex $v \notin \mathsf{HI} \cup \mathsf{HO}$ without any in-neighbor from $V(D) \setminus \mathsf{HI}$ since the proof of the second one is symmetric.

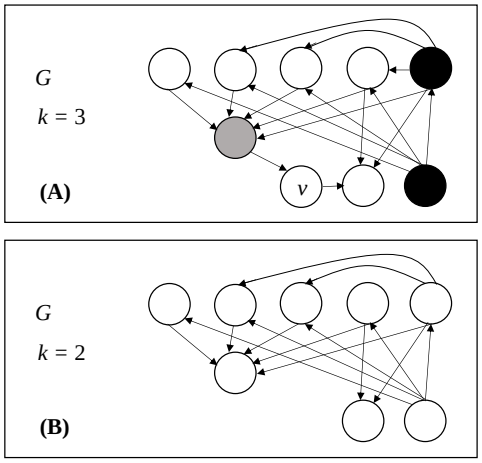


Figure 7.2: (A) An instance of PVDS. The gray vertex belongs to HI, and the black vertices belong to HO. (B) An application of Rule 7.2.6.

In this case, by Assumption 7.2.1, if there exists a solution S, it does not contain exactly one vertex from HO, which will be a source, and exactly one vertex from HI, which will be the target. Indeed, if S does not contain at least two vertices from HO (HI), then since $|S| \leq k$, the pumpkin $D - S$ will contain at least two vertices with out-degree (in-degree) at least two. Suppose $v \notin S$. Since $v \notin \mathsf{HI} \cup \mathsf{HO}$, from Assumption 1, v is neither the source nor the target of the pumpkin $D - S$. Thus, there exists a vertex $u \in N^-(v)$, such that $u \notin S$. Since $N^-(v) \subseteq \mathsf{HI}$, $u \in \mathsf{HI}$. Since $u \in \mathsf{HI}$ and $u \notin S$, from Assumption 1, u is the target vertex of the pumpkin $D - S$, which is a contradiction because u has an out-neighbor v in $D - S$. □

7.3 The Marking Approach

We are now ready to present our marking approach, handling instances to which Assumption 7.2.1 applies and none of the rules in Section 7.2 is applicable. Let $\mathcal{P}^*$ be the set of connected components in $D - (\mathsf{HO} \cup \mathsf{HI})$ that are directed paths whose internal vertices have in-degree 1 and out-degree 1 in D, and let V^* be the union of the vertex-sets of the paths in $\mathcal{P}^*$. It turns out that by relying on Lemma 7.2.3 and Rule 7.2.3, one can directly bound the number of vertices in $V(D) \setminus V^*$ by $\mathcal{O}(k^3)$, assuming that the input instance is a YES instance (see the proof of Lemma 7.3.5). However, bounding the size of V^* is more tricky, and our marking process is devoted to this cause. In this process, we will mark $\mathcal{O}(k^3)$ vertices from V^*, and prove that because we are handling instances to which Assumption 7.2.1 applies, all of the vertices that are not marked are essentially irrelevant. We will perform

two "rounds" of marking. Roughly speaking, for each pair of vertices in HO (or HI) the first round aims to capture enough vertices of paths that describe the relation between the vertices in this pair, or, more precisely, why one of the vertices of the pair is a "better choice" than the other when one should decide which vertex (from HO) is the source of the pumpkin. However, this round is not sufficient, since some vertices in HO (or HI) have conflicts (independent of the other vertices in $\mathsf{HO} \cup \mathsf{HI}$) relating to the endpoints of the paths in $\mathcal{P}^*$. In the second round of marking, for each vertex in $\mathsf{HO} \cup \mathsf{HI}$, we mark enough vertices from these problematic paths.

First Round of Marking. Towards the performance of the first round, we need the following notations. For each vertex $v \in V(D) \setminus (\mathsf{HI} \cup \mathsf{HO})$, let $\widehat{P}(v)$ denote the connected component in $D - (\mathsf{HI} \cup \mathsf{HO})$ containing v. For each $s \in \mathsf{HO}$, let $\widehat{N}(s)$ denote the set of each out-neighbor $v \in V(D) \setminus (\mathsf{HI} \cup \mathsf{HO})$ of s such that $\widehat{P}(v) \in \mathcal{P}^*$ and the first vertex of (the directed path) $\widehat{P}(v)$ is v. Symmetrically, for each $t \in \mathsf{HI}$, let $\widehat{N}(t)$ denote the set of each in-neighbor $v \in V(D) \setminus (\mathsf{HI} \cup \mathsf{HO})$ of t such that $\widehat{P}(v) \in \mathcal{P}^*$ and the last vertex of $\widehat{P}(v)$ is v. By Rule 7.2.5, $\mathsf{HI} \cap \mathsf{HO} = \emptyset$, and therefore these notations are well defined (i.e., we have not defined $\widehat{N}$ twice for the same vertex). Given $u \in (\mathsf{HI} \cup \mathsf{HO})$, we also denote $\widehat{\mathcal{P}}(u) = \{\widehat{P}(v) \mid v \in \widehat{N}(u)\}$. Observe that the paths in $\widehat{\mathcal{P}}(u)$ are pairwise vertex-disjoint.

Next, we identify enough vertices from paths that capture the relation between each pair of vertices in HO (or HI). For each pair $(s, s') \in \mathsf{HO} \times \mathsf{HO}$, let $\widehat{MK}_P(s, s')$ be an arbitrarily chosen set of minimal size of paths from $\widehat{\mathcal{P}}(s) \setminus \widehat{\mathcal{P}}(s')$ that together contain at least $k+1$ vertices not having s' as an in-neighbor. In this context, observe that only the last vertex on a path in $\widehat{\mathcal{P}}(s) \setminus \widehat{\mathcal{P}}(s')$ can have s' as an in-neighbor. In this case, the path must contain at least two vertices (since its first vertex cannot have s' as an in-neighbor), and while we insert the entire path into $\widehat{MK}_P(s, s')$, its last vertex is not "counted" when we aim to obtain at least $k+1$ vertices not having s' as an in-neighbor. If there are not enough paths to obtain at least $k+1$ such vertices, let $\widehat{MK}_P(s, s') = \widehat{\mathcal{P}}(s) \setminus \widehat{\mathcal{P}}(s')$. Symmetrically, for each pair $(t, t') \in \mathsf{HI} \times \mathsf{HI}$, let $\widehat{MK}_P(t, t')$ be an arbitrarily chosen minimal set paths from $\widehat{\mathcal{P}}(t) \setminus \widehat{\mathcal{P}}(t')$ that together contain at least $k+1$ vertices not having t' as an out-neighbor. If there are not enough paths, let $\widehat{MK}_P(t, t') = \widehat{\mathcal{P}}(t) \setminus \widehat{\mathcal{P}}(t')$.

Finally, given a pair $(v, v') \in (\mathsf{HO} \times \mathsf{HO}) \cup (\mathsf{HI} \times \mathsf{HI})$, let $\widehat{MK}(v, v')$ denote the union of the vertex-sets of the paths in $\widehat{MK}_P(v, v')$. We have the following claim.

Lemma 7.3.1. *For each pair $(v, v') \in (\mathsf{HO} \times \mathsf{HO}) \cup (\mathsf{HI} \times \mathsf{HI})$, $|\widehat{MK}(v, v')| \leq 3(k+1)$.*

Proof. For each path inserted into $\widehat{MK}_P(v, v')$, at most one vertex is not counted in

order to reach the desired size $k+1$ (and at least one vertex is counted in order to reach this size). Once we reach the threshold $k+1$ (after marking at most $2(k+1)$ vertices), we may need to add $k+1$ additional vertices, since we insert entire vertex-sets of paths in $\widehat{\mathcal{P}}(s) \setminus \widehat{\mathcal{P}}(s')$, and by Rule 7.2.3, each such path may contain at most $k+2$ vertices. □

Second Round of Marking. We proceed to the second round of marking. For this purpose, we need the following notation. For each $u \in \mathsf{HI} \cup \mathsf{HO}$, let $\widetilde{MK}_P(u)$ denote the set of each directed path in $\mathcal{P}^*$ whose first and last vertices are both neighbors of u.

Reduction Rule 7.3.1. *If there exists $u \in \mathsf{HI} \cup \mathsf{HO}$ such that $|\widetilde{MK}_P(u)| > k+1$, delete u from D and decrease k by 1.*

Lemma 7.3.2. *Reduction Rule 7.3.1 is safe.*

Proof. To prove that the rule is correct, it is sufficient to show that if there exists a solution S, it contains u. Indeed, if S does not contain u, then it must contain at least one vertex from each path in $\widetilde{\mathcal{P}}(u)$, except perhaps one path, since together with u each such path forms a cycle and these cycles intersect only at the vertex u. Since $|\widetilde{\mathcal{P}}(u)| > k+1$ and the paths in $\widetilde{\mathcal{P}}(u)$ are vertex-disjoint, the claim follows. □

For each $u \in \mathsf{HI} \cup \mathsf{HO}$, let $\widetilde{MK}(u)$ be the union of the vertex-sets of the paths in $\widetilde{MK}_P(u)$. Since at this point, Rules 7.2.3 and 7.3.1 are not applicable, we have the following lemma.

Lemma 7.3.3. *For each $u \in \mathsf{HI} \cup \mathsf{HO}$, $|\widetilde{MK}(u)| \leq (k+1)(k+2)$.*

The Size of the Kernel. For the sake of abbreviation, we define the following sets.

- $MK_{\mathcal{P}} = (\bigcup_{(u,u') \in (\mathsf{HO} \times \mathsf{HO}) \cup (\mathsf{HI} \cup \mathsf{HI})} \widehat{MK}_P(u,u')) \cup (\bigcup_{u \in \mathsf{HO} \cup \mathsf{HI}} \widetilde{MK}_P(u))$, and
- $MK = (\bigcup_{(u,u') \in (\mathsf{HO} \times \mathsf{HO}) \cup (\mathsf{HI} \cup \mathsf{HI})} \widehat{MK}(u,u')) \cup (\bigcup_{u \in \mathsf{HO} \cup \mathsf{HI}} \widetilde{MK}(u))$.

By Lemmas 7.3.1 and 7.3.3, and since Rule 7.1.1 is not applicable, we bound $|MK|$ as follows.

Lemma 7.3.4. $|MK| \leq 2 \cdot (3(k+1)^3 + (k+1)^2(k+2)) \leq 8(k+2)^3$.

Let V^R denote the set of unmarked vertices in V^*, i.e., $V^* \setminus MK$. We construct the graph D' by removing from D all of the vertices in V^R, adding a set N_{k+2} of $k+2$ new vertices, and for each of the new vertices, adding an edge from each vertex in HO as well as an edge to each vertex in HI. If $V(D')$ contains at most $2k+4$ vertices, add to it one-by-one a vertex-set of a path in $\mathcal{P}^*$ until its size becomes at least $2k+5$ (by Rule 7.2.3, the size will not exceed $3k+6$, and because $|V(D)| \geq k^3$, we will reach the desired size).

Lemma 7.3.5. *If $|V(D')| > 30(k+2)^3$, (D', k) is a* No *instance of* PVDS.

Proof. If there exists a solution S, the pumpkin $D'-S$ contains at most $2(k+1)|S| \leq 2(k+1)k$ vertices with at least one neighbor in $S \setminus (\mathsf{HI} \cup \mathsf{HO})$. Moreover, by Lemma 7.2.3, the set C associated with $D'-S$ contains at most $4(k+1)|S| \leq 4(k+1)k$ vertices with at least one neighbor in S. Let Z be the set of vertices of these two types. Then, $D'-(S \cup Z \cup \{s,t\})$ is a collection of paths. Among these paths, there are at most $k+2$ paths that are isolated vertices corresponding to the set N_{k+2} of vertices added to D' at its construction. There are at most $2|Z| \leq 12(k+1)k$ additional paths that are not paths in $\mathcal{P}^*$. By Rule 7.2.3, these additional paths together contain at most $12(k+1)k(k+2)$ vertices. Moreover, from the union of the vertex-sets of paths in $\mathcal{P}^*$, D' contains only vertices from MK, and by Lemma 7.3.4, $|MK| \leq 8(k+2)^3$. Summing up, we get that $|V(D)'| \leq (k+2) + 12(k+1)k(k+2) + 8(k+2)^3 + |Z| + |S| + 2 \leq 30(k+2)^3$. □

Correctness. Finally, Theorem 7.0.1 follows from Lemma 7.3.5 and the two lemmas below.

Lemma 7.3.6. *If (D, k) is a* Yes *instance then (D', k) is a* Yes *instance.*

Proof. Let S be a solution to (D, k). Let s and t be the source and sink, respectively, of the pumpkin $D-S$. By Assumption 7.2.1, $s \in \mathsf{HO}$ and $t \in \mathsf{HI}$. Let $S' = S \cap V(D')$. Note that each path in $\mathcal{P}^*$ must either have its vertex-set contained in S or it must be a path from an out-neighbor of s to an in-neighbor of t which belongs to the pumpkin $D-S$ (rather than only a subpath of a path from an out-neighbor of s to an in-neighbor of t which belongs to this pumpkin). Moreover, the set N_{k+2} of vertices added to D' at its construction are vertices of in-degree one and out-degree one, where s is an in-neighbor and t is an out-neighbor. Therefore, $D'-S'$ is a pumpkin. □

Lemma 7.3.7. *If (D', k) is a* Yes *instance then (D, k) is a* Yes *instance.*

Proof. Let S be a solution to (D', k). Let s and t be the source and target, respectively, of the pumpkin $D' - S$. Because of the set N_{k+2} of $k+2$ vertices added to D' at its construction, and since $|S| \leq k$, $s \in \mathsf{HO}$ and $t \in \mathsf{HI}$. Moreover, by the definition of HO and HI, $(\mathsf{HO} \cup \mathsf{HI}) \setminus \{s, t\} \subseteq S$. We can also assume that S does not contain any vertex added to D' at its construction since by removing such a vertex from S, we still have a pumpkin. Our goal will be to show that S is also a solution to (D, k), which will imply that the lemma is correct. To this end, we will show that $D - S$ is a pumpkin with source s and sink t.

First, note that we can assume that in $D - S$ there exists a path from s to t. Indeed, if this is not true, then $D' - S$ consists only of s, t and newly added vertices. That is, $V(D')$ contains at most $2k + 4$ vertices, which contradicts its construction. By the definition of $\mathcal{P}^*$, each path in $\mathcal{P}^*$ has only internal vertices that have in-degree 1 and out-degree 1 in D , and its endpoints can only be adjacent to vertices in $\mathsf{HI} \cup \mathsf{HO}$ and in the path itself. Thus, to prove the lemma, it is sufficient to show that for each path in $\mathcal{P}^* \setminus MK_{\mathcal{P}}$, its first vertex has s as an in-neighbor, its last vertex has t as an out-neighbor, and if it contains at least two vertices, its first vertex is not a neighbor of t and its last vertex is not a neighbor of s.

Consider some path $P \in \mathcal{P}^* \setminus MK_{\mathcal{P}}$. First suppose, by way of contradiction, that the first vertex v of P does not have s as an in-neighbor. Because Rule 7.2.6 is not applicable, v has at least one in-neighbor $s' \in \mathsf{HO}$. Thus, since $v \notin MK$, $MK(s', s)$ contains at least $k + 1$ vertices that are not out-neighbors of s and such that each of them belongs to a path in $\mathcal{P}^*$ whose first vertex is not an out-neighbor of s. The vertices in $MK(s', s)$ belong to D'. Since $|S| \leq k$, at least one of these vertices, say some u, should belong to the pumpkin $D' - S$. However, in $D' - ((\mathsf{HI} \cup \mathsf{HO}) \setminus \{s\})$, which is a supergraph of $D' - S$, u cannot be reached from s, which contradicts the fact that $D' - S$ is a pumpkin. Symmetrically, it is shown that the last vertex of P has t as an out-neighbor.

Now assume that P contains at least two vertices. Suppose, by way of contradiction, that the first vertex of P has t as a neighbor. We have already shown that the last vertex of P is also a neighbor of t, and therefore $P \in \widetilde{MK}_P(t)$. However, $\widetilde{MK}_P(t) \subseteq MK_{\mathcal{P}}$, which contradicts the fact that $P \in \mathcal{P}^* \setminus MK_{\mathcal{P}}$. Symmetrically, it is shown that the last vertex of P does not have s as a neighbor, concluding the proof of the lemma. □

Chapter 8

Faster FPT Algorithms for Deletion to Out-Tree, Out-Forest and Pumpkin

Recall the definitions of out-tree, out-forest and pumpkin from Chapter 6. In this chapter, we complement the kernelization results of Mnich and van Leeuwen [164] by designing faster FPT algorithms for the Out-Forest, Out-Tree and Pumpkin Deletion problems. In particular, we give FPT algorithms for OFVDS, OTVDS and PVDS (recall these definitions from Chapter 6) with running times $\mathcal{O}^\star((1+\sqrt{3})^k) = \mathcal{O}^\star(2.732^k)$, $\mathcal{O}^\star(2.562^k)$ and $\mathcal{O}^\star(2.562^k)$ respectively.

Problem	**Deterministic**	**Randomized**
UFVS	3.619^k	3^k
OFVDS	$(1+\sqrt{3})^k$	-
TDS	5^k	-
OTVDS	2.562^k	-
PVDS	2.562^k	-

Table 8.1: Problems with the best known deterministic and randomized FPT running times.

An underlying undirected graph for both an out-forest and a out-tree does not contain a cycle and thus they are very close to forest and tree, respectively. Thus, it is imperative on us to compare the running time of our FPT algorithms for OFVDS and OTVDS with UFVS and Tree Deletion Set (TDS), respectively. The best known deterministic and randomized algorithms for UFVS run in times

$\mathcal{O}^\star(3.619^k)$ and $\mathcal{O}^\star(3^k)$, respectively [132, 70]. It is a well-known open problem in the area whether one can achieve a deterministic algorithm for UFVS matching the running time of the randomized algorithm. On the other hand the best known algorithm for TDS runs in time $\mathcal{O}^\star(5^k)$ [183]. In contrast to these results, we obtain deterministic algorithms for OFVDS and OTVDS with running times $\mathcal{O}^\star((1+\sqrt{3})^k) = \mathcal{O}^\star(2.732^k)$ and $\mathcal{O}^\star(2.562^k)$ respectively. The main observation which enables us to design all our FPT algorithms is the following:

> Every vertex, except one (in the case of PVDS), in an out-tree, an out-forest or pumpkin has in-degree at most one; in pumpkin every vertex except s and t has in-degree one and out-degree one. Thus for any vertex v, amongst v and any two of its in-neighbors, say x and y, one of the vertices must belong to the deletion set. We will call this 3-*hitting set* property. This results is a 3 way branching. When such a structure does not exist, in the case of Out-forest and Out-tree vertex deletion set problems, each vertex has in-degree at most 1, and in the case of the Pumpkin vertex deletion set problem, each vertex, except for one, has indegree at most 1. In the former case, observe that, when each vertex in a digraph has in-degree at most 1, then the digraph is a disjoint collection of out-trees and directed cycles. In the later case, if the input digraph is a Yes instance, then the resulting digraph is a disjoint collection of a pumpkin, out-trees and directed cycles. In either case, the corresponding problem on the resulting digraph can be solved in polynomial time. This already gives us deterministic algorithms with running time $\mathcal{O}^\star(3^k)$ for OFVDS, OTVDS, and PVDS. We exploit this hitting set property together with the fact that out-forest, out-tree and pumpkin are all almost acyclic graphs, to design FPT algorithms faster than $\mathcal{O}^\star(3^k)$.

Here we would also like to mention the corresponding *extension* problems of the problems considered in the article, which we define as follows.

OFVDS/ OTVDS/ PVDS- Extension	**Parameter:** k
Input: A directed graph D, $X \subseteq V(D)$ and a positive integer k.	
Question: Is there a set $S \subseteq V(D) \setminus X$ of size at most k such that $F = D-(S \cup X)$ is an out-forest/ out-tree/ pumpkin?	

Observe that if we have FPT algorithms for OFVDS/ OTVDS/ PVDS then

to solve instances (D, X, k) of OFVDS/ OTVDS/ PVDS- Extension problems, we can run the FPT algorithms of Out-forest/Out-tree/Pumpkin Vertex Deletion Set problems on the instance $(D - X, k - |X|)$. Thus, FPT algorithms for Out-forest/Out-tree/Pumpkin Vertex Deletion Set problems imply FPT algorithms for Out-forest/Out-tree/Pumpkin Vertex Deletion Set - Extension problems. Combining this observation with Theorem 2 of [94] (which roughly states that if there an FPT algorithm for a vertex-deletion extension problem with running time $c^k n^{\mathcal{O}(1)}$ then there is an exact algorithm for the corresponding vertex-deletion problem with running time $\left(2 - \frac{1}{c}\right)^n n^{\mathcal{O}(1)}$), we get exact algorithms with running time $\mathcal{O}^*(1.634^n)$, $\mathcal{O}^*(1.610^n)$ and $\mathcal{O}^*(1.610^n)$ respectively, for OFVDS/ OTVDS/ PVDS.

As mentioned before, for all the problems considered in this chapter, an $\mathcal{O}^*(3^k)$ algorithm is trivial. The non-trivial running times are obtained by choosing an appropriate vertex to branch on. The order of the branching rules imposes a lot of structure in the cases that appear later. This structure is exploited to achieve the target running time.

In all our algorithms, if there exists a pair of vertices u, v in the input digraph such that both (u, v) and (v, u) belong to the arc set of the digraph, then observe that at least one of u and v belong to the solution. Thus, we branch on such vertex pairs, reducing the budget by one in each branch and stopping whenever the budget is zero or negative. Henceforth, without loss of generality, we assume that in the input digraph, for any pair u, v of vertices there is at most of arcs (u, v) or (v, u).

8.1 FPT Algorithm for Pumpkin Vertex Deletion Set

In this section, we give a branching algorithm for Pumpkin Vertex Deletion Set. Let (D, k) be an instance of PVDS. The algorithm starts by guessing the source vertex s and the sink vertex t in the graph obtained after the deletion of the solution vertices. There are $O(|V(D)|^2)$ choices for s and t. After this guesswork, we want to solve the problem where we are given a digraph D, a positive integer k and two vertices s and t, and the question is does there exist a set S of the vertices of D of size at most k such that $D - S$ is a pumpkin with s as the source vertex and t as the sink vertex. We call this new problem Restricted Pumpkin Vertex Deletion Set (RPVDS) and in the rest of this section we develop an algorithm for solving it. This algorithm together with the guessing step will give an FPT

algorithm for PVDS. Formally, RPVDS is defined as follows.

Restricted Pumpkin Vertex Deletion Set (RPVDS) **Parameter:** k
Input: A directed graph D, two distinct vertices $s, t \in V(D)$ and a positive integer k.
Question: Is there a set $S \subseteq V(D)$ of size at most k such that $D - S$ is a pumpkin with s as the source vertex and t as the sink vertex?

Broadly speaking, our strategy is to branch on vertices having in-degree at least 2 or out-degree at least 2. The reduction rules and branching rules are so designed that when none of them is applicable, all vertices in the resulting digraph, except for s and t, have in-degree exactly 1 and out-degree exactly 1 and, s has in-degree 0 and t has out-degree 0. Such an instance becomes trivial to solve. To achieve this trivial instance, the algorithm systematically deals with vertices that do not satisfy the constraints of this trivial instance.

We now give the formal description of the algorithm. The measure μ that will be used to bound the depth of the search tree of our branching algorithm is the solution size, that is, $\mu(D, k, s, t) = k$.

With a slight abuse of notation, in the following, during the application of any reduction/branching rule we will refer to (D, k, s, t) as the instance that is reduced with respect to the rules in higher preference order. We first list the reduction rules used by the algorithm.

Reduction Rule 8.1.1. *If $k < 0$, return (D, k, s, t) is a* No *instance.*

Reduction Rule 8.1.2. *If $k = 0$ and D is not a pumpkin with source s and sink t, return (D, k, s, t) is a* No *instance.*

Reduction Rule 8.1.3. *If $k \geq 0$ and D is a pumpkin with source s and sink t, return (D, k, s, t) is a* Yes *instance.*

Reduction Rule 8.1.4. *If there exists $v \in V(D) \setminus \{s\}$ such that $d^-(v) = 0$ then delete v from D and decrease k by 1. That is, the resulting instance is $(D - \{v\}, k - 1, s, t)$.*

Reduction Rule 8.1.5. *If there exists $v \in V(D) \setminus \{t\}$ such that $d^+(v) = 0$ then delete v from D and decrease k by 1. That is, the resulting instance is $(D - \{v\}, k - 1, s, t)$.*

Reduction Rule 8.1.6. *If there exists $v \in V(D)$ such that $s \in N^+(v)$ then delete v from D and decrease k by 1. That is, the resulting instance is $(D - \{v\}, k - 1, s, t)$.*

Reduction Rule 8.1.7. *If there exists $v \in V(D)$ such that $t \in N^-(v)$ then delete v from D and decrease k by 1. That is, the resulting instance is $(D - \{v\}, k-1, s, t)$.*

Reduction Rule 8.1.8. *If C is a weakly connected component of D such that either $s \notin V(C)$ or $t \notin V(C)$, then the resulting instance is $(D - V(C), k - |C|, s, t)$.*

Reduction Rule 8.1.9. *If $s \notin V(D)$ or $t \notin V(D)$, return (D, k, s, t) is a* No *instance.*

It is easy to see that reduction rules 8.1.1 to 8.1.9 are safe and can be applied in polynomial time.

We now describe the branching rules used by the algorithm. The algorithm branches on some vertex based on its in-degree and/or out-degree as per one of the 5 branching rules described later. Before giving the details of the branching rules, we first mention the invariants maintained by the algorithm after the exhaustive application of each of the branching rules.

More precisely, the invariant maintained at the end of branching rule i is a condition which always holds when none of the reduction rules and branching rules 1 to i are applicable.

Branching Rule 1: For all $v \in V(D) \setminus \{s, t\}$, if $d^-(v) \geq 2$, then $s \notin N^-(v)$. Symmetrically, if $d^+(v) > 1$, then $t \notin N^+(v)$.

Branching Rule 2: For all $v \in V(D) \setminus \{s, t\}$, $d^-(v) \leq 2$. Symmetrically, $d^+(v) \leq 2$.

Branching Rule 3: For all $v \in V(D) \setminus \{s, t\}$, if $d^-(v) = 2$, then $d^+(v) \leq 1$. Symmetrically, if $d^+(v) = 2$, then $d^-(v) \leq 1$.

Branching Rule 4: For all $v \in V(D) \setminus \{s, t\}$, if $d^-(v) = 2$ and $d^+(v) = 1$, then $N^+(v) = \{t\}$. Symmetrically, if $d^+(v) = 2$ and $d^-(v) = 1$, then $N^-(v) = \{s\}$.

Branching Rule 5: For all $v \in V(D) \setminus \{s, t\}$, if $d^-(v) = 2$ then $d^+(v) = 0$. Symmetrically, if $d^+(v) = 2$, then $d^-(v) = 0$.

We now give the description of the branching rules. From the description of the branching rules it is easy to see that each branching rule is exhaustive.

Branching Rule 1 If there is $v \in V(D)$ such that $d^-(v) \geq 2$ (or $d^+(v) \geq 2$) and $s \in N^-(v)$ (or $t \in N^+(v)$), then let u be the other in-neighbour (or out-neighbour) of v. In this case the algorithm branches as follows.

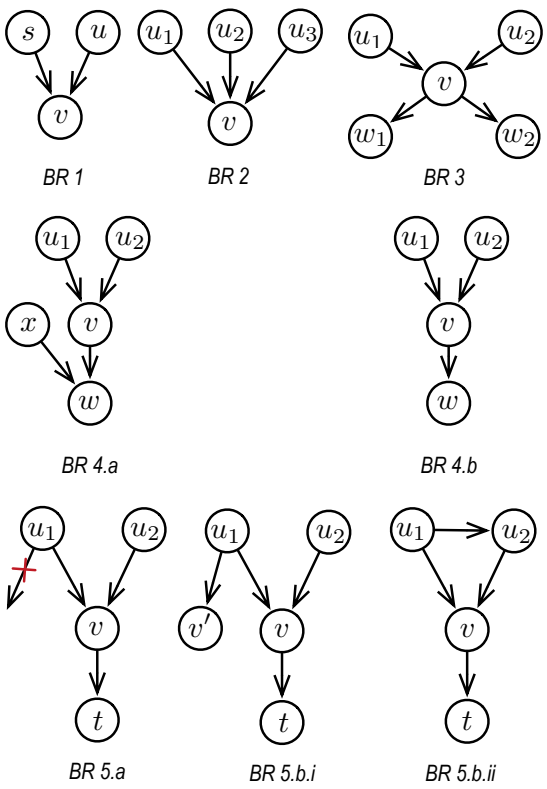


Figure 8.1: The structures in the branching rules of PVDS

- When v belongs to the solution, then the resulting instance is $(D - \{v\}, k - 1, s, t)$.
- When v does not belong to the solution, then u must belong to the solution as otherwise, v, u and s (or t) will belong to the pumpkin obtained after the solution vertices are deleted, which is not possible. Therefore, the resulting instance is $(D - \{u\}, k - 1, s, t)$.

The branching vector for this rule is $(1, 1)$.

Branching Rule 2 If there is $v \in V(D)$ such that $d^-(v) \geq 3$ (or $d^+(v) \geq 3$), $v \neq t$ (or $v \neq s$), then let u_1, u_2, u_3 be some three distinct in-neighbors (or out-neighbors) of v. Note that none of u_1, u_2, u_3 is the same as s (or t), as otherwise branching rule 1 would be applicable. In this case, the algorithm branches as follows.

- When v belongs to the solution, the resulting instance is $(D - \{v\}, k - 1, s, t)$.
- When v does not belong to the solution, then note that at least 2 of u_1, u_2 and u_3 must belong to the solution. Thus, the algorithm further branches as follows.
  - When u_1 and u_2 belong to the solution, the resulting instance is $(D - \{u_1, u_2\}, k - 2, s, t)$.
  - When u_2 and u_3 belong to the solution, the resulting instance is $(D - \{u_2, u_3\}, k - 2, s, t)$.
  - When u_1 and u_3 belong to the solution, the resulting instance is $(D - \{u_1, u_3\}, k - 2, s, t)$.

The branching vector for this rule is $(1, 2, 2, 2)$.

Branching Rule 3 If there is $v \in V(D) \setminus \{s, t\}$ such that $d^-(v) = 2$ and $d^+(v) = 2$, then let u_1, u_2 be the in-neighbors of v and w_1, w_2 be the out-neighbors of v. Observe that neither u_1 nor u_2 is the same as t, as otherwise branching rule 1 would be applicable. Similarly, neither w_1 nor w_2 is the same as s. In this case, the algorithm branches as follows.

- When v belongs to the solution, the resulting instance is $(D - \{v\}, k - 1, s, t)$.

- When v does not belong to the solution, then note that at least one of u_1 or u_2 and, at least one of w_1 or w_2 must belong to the solution. Thus, the algorithm further branches as follows.

  - When u_1 and w_1 belong to the solution, the resulting instance is $(D - \{u_1, w_1\}, k-2, s, t)$.
  - When u_1 and w_2 belong to the solution, the resulting instance is $(D - \{u_1, w_2\}, k-2, s, t)$.
  - When u_2 and w_1 belong to the solution, the resulting instance is $(D - \{u_2, w_1\}, k-2, s, t)$.
  - When u_2 and w_2 belong to the solution, the resulting instance is $(D - \{u_2, w_2\}, k-2, s, t)$.

The branching vector for this rule is $(1, 2, 2, 2, 2)$.

Branching Rule 4 If there is $v \in V(D) \setminus \{s, t\}$ such that $d^-(v) = 2$ (or $d^+(v) = 2$), $d^+(v) = 1$ (or $d^-(v) = 1$) and the unique out-neighbor (or in-neighbor), say w of v, is not t (or s), then let u_1, u_2 be the two in-neighbors (or out-neighbors) of v. Observe that none of u_1 or u_2 is the same as s. The algorithm considers the following cases depending on the in-degree (or out-degree) of w.

Case 4.a. If $d^-(w) = 2$ (or $d^+(w) = 2$), then let x be the other in-neighbor (or out-neighbor) of w different from v. Here x may or may not be equal to u_1 or u_2. In this case, the algorithm branches as follows.

- When v belongs to the solution, the resulting instance is $(D - \{v\}, k-1, s, t)$.
- When v does not belong to the solution, then note that w does not belong to the solution (because if it does, then v must be a sink vertex and hence $v = t$, which is not true). Since $w \neq t$, and both v and w do not belong to the solution, x must belong to the solution. Therefore, the resulting instance is $(D - \{x\}, k-1, s, t)$.

The branching vector in this case is $(1, 1)$.

Case 4.b. If $d^-(w) = 1$ (or $d^+(w) = 1$), the algorithm branches as follows.

- When v belongs to the solution, then delete v from the graph. In the resulting graph $d^-(w) = 0$. Since $w \neq s$, reduction rule 8.1.4 is applicable. Therefore, the resulting instance is $(D - \{v, w\}, k - 2, s, t)$.
- When v does not belong to the solution, then at least one of u_1 or u_2 must belong to the solution. Hence, the algorithm branches further into 2 cases. In the first case, the resulting instance is $(D - \{u_1\}, k - 1, s, t)$ and in the second case the resulting instance is $(D - \{u_2\}, k - 1, s, t)$.

The branching vector in this case is $(2, 1, 1)$.

Branching Rule 5 If there is $v \in V(D) \setminus \{s, t\}$ such that $d^-(v) = 2$ (or $d^+(v) = 2$), $d^+(v) = 1$ (or $d^-(v) = 1$) and t (or s) is the unique out-neighbor (or in-neighbor) of v, then let u_1, u_2 be the two in-neighbors (or out-neighbors) of v. Observe that none of u_1 or u_2 is same as s (or t), otherwise branching rule 1 would be applicable. The algorithm considers the following cases based on the out-degree (or in-degree) of u_1 and u_2.

Case 5.a. If either $d^+(u_1) = 1$ or $d^+(u_2) = 1$ ($d^-(u_1) = 1$ or $d^-(u_2) = 1$), then without loss of generality assume that $d^+(u_1) = 1$ (or $d^-(u_1) = 1$).

Observe that u_1 is distinct from t (or s) because u_1 is an in-neighbor (or out-neighbor) of v and t (or s) is an out-neighbor of v. In this case, the algorithm branches as follows.

- When u_1 belongs to the solution, then delete u_1 from the graph. Thus the resulting instance is $(D - \{u_1\}, k - 1, s, t)$.
- When u_1 does not belong to the solution, since $d^+(u_1) = 1$, and $N^+(u_1) = \{v\}$, v does not belong to the solution. Since $v \neq t$, and $u_2 \in N^-(v)$, u_2 belongs to the solution. Thus, the resulting instance is $(D - \{u_2\}, k - 1, s, t)$.

The branching vector for this case is $(1, 1)$.

Case 5.b. If $d^+(u_1) = 2$ and $d^+(u_2) = 2$ ($d^-(u_1) = 2$ and $d^-(u_1) = 2$), let v' be the other out-neighbor (or in-neighbor) of u_1. In this case, the algorithm considers the following sub-cases.

Sub-case 5.b.i. If v' is different from u_2 and t, then the algorithm branches as

follows.

- When v belongs to the solution, the resulting instance is $(D - \{v\}, k-1, s, t)$.
- When v does not belong to the solution, then observe that at least one of u_1 and u_2 must belong to the solution. Thus, the algorithm branches as follows.
  - When u_1 belongs to the solution, the resulting instance is $(D - \{u_1\}, k-1, s, t)$.
  - When u_1 does not belong to the solution, then u_2 must belong to the solution. Also v' must belong to the solution (because u_1 is distinct from s). Therefore, the resulting instance is $(D - \{u_2, v'\}, k-2, s, t)$.

The branching vector for this case is $(1, 1, 2)$.

Sub-case 5.b.ii. If v' is the same as t, the algorithm branches as follows.

- When v belongs to the solution, the resulting instance is $(D - \{v\}, k-1, s, t)$.
- When v does not belong to the solution then u_1 must belong to the solution (because $u_1 \neq s$). Therefore, the resulting instance is $(D - \{u_1\}, k-1, s, t)$.

The branching vector for this case is $(1, 1)$.

Sub-case 5.b.iii. If v' is the same as u_2, then we know that $d^+(u_2) = 2$, and therefore, one of case 5.a.i or case 5.a.ii would be applicable.

This ends the description of the branching rules. In the upcoming lemma, we show that when all the reduction rules and branching rules have been applied exhaustively, that is when none of them is no longer applicable, all the vertices of the resulting instance, possibly except for s and t, have in-degree exactly 1 and out-degree exactly 1. In the later lemma we show that such an instance can be solved in polynomial time. Thus, after the exhaustive application of the reduction rules and the above mentioned branching rules, the algorithm uses the procedure described in Lemma 8.1.2 to solve the instance.

Lemma 8.1.1. *Let (D, k, s, t) be the instance where none of the above-mentioned reduction rules or branching rules are applicable. Then, for all $v \in V(D) \setminus \{s, t\}$, $d^-(v) = 1$, $d^+(v) = 1$, $d^-(s) = 0$ and $d^+(t) = 0$.*

Proof. Since reduction rules 8.1.6 and 8.1.7 are no longer applicable, we have $d^-(s) = 0$ and $d^+(t) = 0$. Also since reduction rules 8.1.4 and 8.1.5 are not applicable, $d^-(v) > 0$ and $d^+(v) > 0$, for all $v \in V(D) \setminus \{s, t\}$. To show that for any vertex $v \in V(D) \setminus \{s, t\}$, $d^-(v) = 1$ we proceed as follows.

Let v be some vertex in $V(D)\setminus\{s, t\}$. For the sake of contradiction, let $d^-(v) > 1$. If $d^-(v) > 1$ and $s \in N^-(v)$, then branching rule 1 would be applicable. Thus, we can safely assume that $s \notin N^-(v)$.

We split the situation that $d^-(v) > 1$, into 2 cases as follows.

- If $d^-(v) \geq 3$, then branching rule 2 would be applicable.
- Otherwise $1 < d^-(v) \leq 2$, that is, $d^-(v) = 2$. We split this situation into the following 3 exhaustive cases.
  - If $d^+(v) \geq 2$, then branching rule 3 would be applicable.
  - If $d^+(v) = 1$, then based on whether the unique out-neighbour of v is t or not, either branching rule 4 or branching rule 5 would be applicable.
  - If $d^+(v) = 0$, then reduction rule 8.1.5 would be applicable.

Since, none of the reduction rules or branching rules are applicable, we conclude that for all $v \in V(D) \setminus \{s, t\}$, $d^-(v) = 1$. Using the same case analysis we can show that for any vertex $v \in V(D) \setminus \{s, t\}$, $d^+(v) = 1$. □

Lemma 8.1.2. *Let (D, k, s, t) be an instance of* Restricted Pumpkin Vertex Deletion Set *such that for all $v \in V(D) \setminus \{s, t\}$, $d^-(v) \leq 1$, $d^+(v) \leq 1$ and, $d^-(s) = 0$ and $d^+(t) = 0$. Then the instance (D, k, s, t) can be solved in $\mathcal{O}(n^{\mathcal{O}(1)})$ time, where $n = |V(D)|$.*

Proof. Since reduction rule 8.1.8 and 8.1.9 are not applicable, D is weakly connected and both s and t belong to D. Since, for all $v \in V(D) \setminus \{s, t\}$, $d^-(v) = 1$, $d^+(v) = 1$ and, $d^-(s) = 0$ and $d^+(t) = 0$, and observe that (D, k, s, t) is a Yes instance if and only if D is a pumpkin with s as the source vertex, t as the sink vertex and $k \geq 0$. □

In the next lemma, we formally prove the correctness of our algorithm.

Lemma 8.1.3. *The algorithm presented for* Restricted Pumpkin Vertex Deletion Set *is correct.*

Branching Rule (BR)	Case/Sub-Case	Branch Vector	c^{μ}
BR 1		$(1,1)$	2^{μ}
BR 2		$(1,2,2,2)$	2.30278^{μ}
BR 3		$(1,2,2,2,2)$	2.56156^{μ}
BR 4	a b	$(1,2,1)$ $(2,1,1)$	2.41422^{μ} 2.41422^{μ}
BR 5	a b.i b.ii	$(1,1)$ $(1,1,2)$ $(1,1)$	2^{μ} 2.41422^{μ} 2^{μ}

Table 8.2: The branch vectors and their corresponding running times for RPVDS.

Proof. Let $I = (D, k, s, t)$ be an instance of RPVDS. We prove the correctness of the algorithm by induction on $\mu = \mu(I) = k$. The base case occurs in one of the following cases.

- If one of s or t does not belong to $V(D)$, the algorithm correctly concludes that (D, k, s, t) is a NO instance from reduction rule 8.1.8.
- If $\mu <= 0$, the algorithm correctly concludes whether (D, k, s, t) is a yes instance or not by reduction rules 8.1.1 to 8.1.3.
- If $\mu >= 0$ and D is a pumpkin, the algorithm correctly concludes that (D, k, s, t) is a YES instance.
- If $\mu \geq 0$ and for all $v \in V(D) \setminus \{s, t\}$, $d^-(v) = 1$, $d^+(v) = 1$ and, $d^-(s) = 0$, $d^+(t) = 0$, then from Lemma 8.1.1 the algorithm solves the instance correctly.

By induction hypothesis we assume that for all $\mu \leq l$, the algorithm is correct. We will now prove that the algorithm is correct when $\mu = l + 1$. The algorithm performs one of the following actions. If possible, it applies one of the reduction rules. By the safeness of the reduction rules we either correctly conclude that I is a YES/NO instance or produce an equivalent instance I' with $\mu(I') \leq \mu(I)$. If $\mu(I') < \mu(I)$, then by induction hypothesis and safeness of the reduction rules the algorithm correctly decides if I is a YES instance or not. Otherwise, $\mu(I') = \mu(I)$. If none of the reduction rules are applicable then the algorithm applies the first applicable Branching Rule. If some branching rule is applicable then, since μ decreases in each branch by at least one, by induction hypothesis the algorithm correctly concludes that I is a YES/ NO instance. If none of the branching rules is applicable, then from Lemma 8.1.2, if $I = (D, k, s, t)$, then for all $v \in V(D) \setminus \{s, t\}$, $d^-(v) = 1$, $d^+(v) = 1$

and, $d^-(s) = 0$, $d^+(t) = 0$. Thus, in this situation we handle a base case correctly and hence, we conclude that the algorithm always outputs the correct answer. □

Next, we analyze the running time of our algorithm.

Theorem 8.1.1. *The presented algorithm solves* Restricted Pumpkin Vertex Deletion Set *in time* $\mathcal{O}^\star(2.562^k)$.

Proof. Observe that reduction rules 8.1.1 to 8.1.9 can be applied in time polynomial in the input size and are never applied more than polynomial number of times. Also, at each of the branches we spend a polynomial amount of time. At each of the recursive calls in a branch, the measure μ decreases by at least by 1. When $\mu \leq 0$, then we are able to solve the instance in polynomial time or correctly conclude that the corresponding branch cannot lead to a solution. At the start of the algorithm $\mu = k$.

The worst-case branching vector for the algorithm is $(1, 2, 2, 2, 2)$ (see Table 14.1). The recurrence for the worst case branching vector is

$$T(\mu) \leq T(\mu - 1) + 4T(\mu - 2)$$

The running time corresponding to the above recurrence relation is $\mathcal{O}^\star(2.562^k)$. □

As mentioned at the starting of the section, given an instance (D, k) of PVDS, one can design an algorithm for PVDS which guesses the source and sink vertices, s and t respectively, of the resulting pumpkin and for each such guess runs the algorithm for RPVDS on (D, k, s, t). Note that (D, k) is a Yes instance of PVDS if and only if (D, k, s, t) is a Yes instance of RPVDS for some guess of s and t. Since there are at most $|V(D)|^2$ choices for the pair s and t, Theorem 8.1.1 gives us the following theorem.

Theorem 8.1.2. Pumpkin Vertex Deletion Set *can be solved in time* $\mathcal{O}^\star(2.562^k)$.

8.2 FPT Algorithm for Out-Tree Vertex Deletion Set

In this section, we give a branching algorithm for Out-Tree Vertex Deletion Set (OTVDS). In the broader picture, this algorithm is in the same spirit as the one

for PVDS with a difference only in the details of the reduction rules and branching rules.

Let (D, k) be an instance of OTVDS. The algorithm starts by guessing the root vertex r of the out-tree obtained after the deletion of the solution vertices. Note that there are $|V(D)|$ choices for r. After this guesswork, we would like to solve the following problem. Given a digraph D, an integer k and a vertex r of the digraph, does there exist a set of at most k vertices whose deletion results in an out-tree with root r. We call this new problem Restricted Out-Tree Vertex Deletion Set and give an FPT algorithm for this problem parameterized by the solution size. Note that the algorithm for this new problem combined with the original guess of the vertex r gives an FPT algorithm for OTVDS. Formally, Restricted Out-Tree Vertex Deletion Set is defined as follows.

Restricted Out-Tree Vertex Deletion Set (ROTVDS) **Parameter:** k
Input: A directed graph D, a vertex $r \in V(D)$ and a positive integer k.
Question: Is there a set $S \subseteq V(D)$ of size at most k such that $D - S$ is an out-tree with r as the root vertex?

We first give an outline of the algorithm for ROTVDS. Let (D, k, r) be an instance of OTVDS. The reduction rules and branching rules for this algorithm are so designed that after the exhaustive application of these rules, all vertices in the resulting instance, except r, have in-degree exactly 1 and the in-degree of r is 0. Such an instance becomes trivial to solve. To achieve this trivial instance, the algorithm systematically deals with vertices that do not satisfy the constraints of this trivial instance.

We now give the formal description of the algorithm. The measure μ that will be used to bound the depth of the search tree of our branching algorithm is the solution size, that is, $\mu(D, k, s, t) = k$. With a slight abuse of notation, in the following, during the application of any reduction/branching rule we will refer to (D, k, s, t) as the instance that is reduced with respect to the rules in higher preference order. We first list the reduction rules used by the algorithm.

Reduction Rule 8.2.1. *If $k < 0$, then (D, k, r) is a* No *instance.*

Reduction Rule 8.2.2. *If $k = 0$ and D is not an out-tree, return (D, k, r) is a* No *instance.*

Reduction Rule 8.2.3. *If $k \geq 0$ and D is an out-tree, return (D, k, r) is a* Yes *instance.*

Reduction Rule 8.2.4. *If there exists $v \in V(D) \setminus \{r\}$ such that $d^-(v) = 0$ then delete v from D and decrease k by 1. That is, the resulting instance is $(D - \{v\}, k - 1, r)$.*

Reduction Rule 8.2.5. *If there exists $v \in V(D)$ such that $r \in N^+(v)$ then delete v from D and decrease k by 1. That is, the resulting instance is $(D - \{v\}, k - 1, r)$.*

Reduction Rule 8.2.6. *If there exists a weakly connected component C not containing r, then delete all the vertices in C. That is, the resulting instance is $(D - V(C), k - |V(C)|, r)$.*

Reduction Rule 8.2.7. *If $r \notin V(D)$, return (D, k, r) is a* No *instance.*

The algorithm applies reduction rules 8.2.1 to 8.2.7 (in order) exhaustively. It is easy to see that reduction rules 8.2.1 to 8.2.7 are safe and can be applied in polynomial time.

We now describe the branching rules used by the algorithm. The algorithm branches on some vertex based on its in-degree and/or out-degree as per one of the 5 branching rules described later. Before giving the details of the branching rules, we first mention the invariants maintained by the algorithm after the exhaustive application of each of the branching rules.

Branching Rule 1: For all $v \in V(D) \setminus \{r\}$, if $d^-(v) \geq 2$, then $r \notin N^-(v)$.

Branching Rule 2: For all $v \in V(D) \setminus \{r\}$, $d^-(v) \leq 2$.

Branching Rule 3: For all $v \in V(D) \setminus \{r\}$, if $d^-(v) = 2$, then $d^+(v) = 0$.

Branching Rule 4: For all $v \in V(D) \setminus \{r\}$, if $d^-(v) = 2$ and $d^+(v) = 0$, then for all $u \in V(D)$, $u \neq v$, if $d^-(u) = 2$ and $d^+(u) = 0$, then v and u have no common in-neighbour.

Branching Rule 5: For all $v \in V(D) \setminus \{r\}$, $d^-(v) \leq 1$.

We now give the description of the branching rules. From the description of the branching rules it is easy to see that each branching rule is exhaustive.

Branching Rule 1 If there exists $v \in V(D)$ such that $d^-(v) \geq 2$ and $r \in N^-(v)$, let u be one of the other in-neighbours of v. In this case, the algorithm branches as follows.

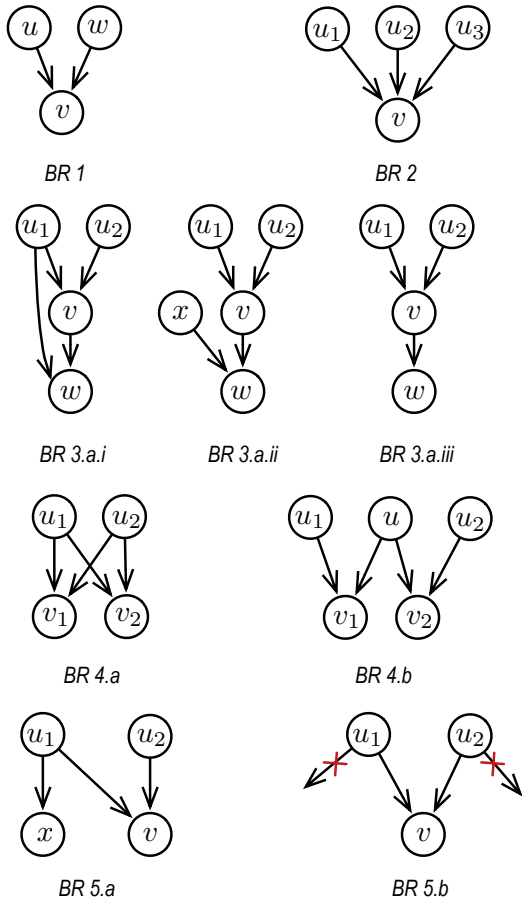


Figure 8.2: The structures in the branching rules of OTVDS

- When v belongs to the solution, the resulting instance is $(D - \{v\}, k-1, r)$.
- When v does not belong to the solution, then u must belong to the solution. Therefore, the resulting instance is $(D - \{u\}, k-1, r)$.

The branching vector for this rule is $(1, 1)$.

Branching Rule 2 (**In-degree at least** 3 **Rule**) If there is $v \in V(D)$ such that $d^-(v) \geq 3$, let u_1, u_2, u_3 be some distinct in-neighbours of v. Note that none of u_1, u_2, u_3 is same as r, as otherwise branching rule 1 would be applicable. In this case, the algorithm branches as follows.

- When v belongs to the solution, the resulting instance is $(D - \{v\}, k-1, r)$.
- When v does not belong to the solution then observe that at least 2 of u_1, u_2, u_3 must belong to the solution. Thus, the algorithm branches as follows.
  - When u_1 and u_2 belong to the solution, the resulting instance is $(D - \{u_1, u_2\}, k-2, r)$.
  - When u_2 and u_3 belong to the solution, the resulting instance is $(D - \{u_2, u_3\}, k-2, r)$.
  - When u_1 and u_3 belong to the solution, the resulting instance is $(D - \{u_1, u_3\}, k-2, r)$.

The branching vector for this rule is $(1, 2, 2, 2)$.

After the exhaustive application of reduction rule 8.2.1 to 8.2.7 and when branching rules 1 and 2 are no more applicable, for all $v \in V(D) \setminus \{r\}$, $1 \leq d^-(v) \leq 2$ and $d^-(r) = 0$.

Branching Rule 3 If $v \in V(D)$ be such that $d^-(v) = 2$ and $d^+(v) \geq 1$, let u_1, u_2 be the two in-neighbours and w be one of the out-neighbours of v. The algorithm considers the following cases depending on the in-degree of w.

Case 3.a. If $d^-(w) = 2$, let x be the other in-neighbour of w. The algorithm further considers the following sub-cases.

Sub-case 3.a.i. If x is the same as u_1 (symmetrically u_2), then the algorithm branches as follows.

- When v belongs to the solution, the resulting instance is $(D - \{v\}, k - 1, r)$.
- When v does not belong to the solution, the algorithm branches on u_1 as follows.
  - When u_1 belongs to the solution, the resulting instance is $(D - \{u_1\}, k - 1, r)$.
  - When u_1 does not belong to the solution, then observe that both u_2 and w must belong to the solution. Thus, the resulting instance is $(D - \{u_2, w\}, k - 2, r)$.

The branching vector for this case is $(1, 1, 2)$.

Sub-case 3.a.ii. Otherwise, x is distinct from u_1 and u_2. In this case, the algorithm branches as follows.

- When v belongs to the solution, the resulting instance is $(D - \{v\}, k - 1, r)$.
- When v does not belong to the solution then observe that at least one of u_1, u_2 and at least one of w, x must belong to the solution. Thus, the algorithm branches as follows.
  - When u_1 and w belongs to the solution, the resulting instance is $(D - \{u_1, w\}, k - 2, r)$.
  - When u_1 and x belongs to the solution, the resulting instance is $(D - \{u_1, x\}, k - 2, r)$.
  - When u_2 and w belongs to the solution, the resulting instance is $(D - \{u_2, w\}, k - 2, r)$.
  - When u_2 and x belongs to the solution, the resulting instance is $(D - \{u_2, x\}, k - 2, r)$.

The branching vector for this case is $(1, 2, 2, 2, 2)$.

Case 3.b. If $d^-(w) = 1$, then observe that w are r are distinct because $d^-(r) = 0$ (as reduction rule 8.2.5 is no longer applicable). In this case, the algorithm branches as follows.

- When v belongs to the solution, then delete v from the graph. Observe that in the resulting graph $d^-(w) = 0$ and hence, reduction rule 8.2.4 would be applicable. Therefore, the resulting instance in this case is $(D-\{v,w\}, k-2, r)$.
- When v does not belong to the solution then branch on u_1 as follows.
  - When u_1 belongs to the solution, the resulting instance is $(D-\{u_1\}, k-1, r)$.
  - When u_1 does not belong to the solution, then u_2 must belong to the solution. Thus the resulting instance is $(D-\{u_2\}, k-1, r)$.

The branching vector in this case is $(2, 1, 1)$.

We now consider the case when there are two vertices of in-degree 2 that have a common in-neighbour.

Branching Rule 4 If $v_1, v_2 \in V(D)$ such that $d^-(v_1) = d^-(v_2) = 2$, $d^+(v_1) = d^+(v_2) = 0$ and there exists at least one common in-neighbour of v_1 and v_2, then the algorithm considers the following sub-cases.

Case 4.a. If v_1 and v_2 have two common in-neighbours, say u_1, u_2, then the algorithm branches as follows.

- When u_1 belongs to the solution, then the resulting instance is $(D-\{u_1\}, k-1, r)$.
- When u_1 does not belong to the solution, then the algorithm further branches as follows.
  - When u_2 belongs to the solution, then the resulting instance is $(D-\{u_2\}, k-1, r)$.
  - When u_2 does not belong to the solution, then observe that both v_1 and v_2 must belong to the solution. Thus the resulting instance is $(D-\{v_1, v_2\}, k-2, r)$.

The branching vector for this case is $(1, 1, 2)$.

Case 4.b. Otherwise, v_1, v_2 have exactly one common in-neighbour, say u, and x_1, x_2 are the other in-neighbours of v_1, v_2 respectively ($x_1 \neq x_2$). In this case, the algorithm branches as follows.

- When u belongs to the solution, the resulting instance is $(D - \{u\}, k-1, r)$.
- When u does not belong to the solution, then observe that at least one of v_1 or x_1, and at least one of v_2 or x_2 must belong to the solution. Therefore, the algorithm branches as follows.
  - When v_1, v_2 belong to the solution, the resulting instance is $(D-\{v_1, v_2\}, k-2, r)$.
  - When v_1, x_2 belong to the solution, the resulting instance is $(D-\{v_1, x_2\}, k-2, r)$.
  - When x_1, v_2 belong to the solution, the resulting instance is $(D-\{x_1, v_2\}, k-2, r)$.
  - When x_1, x_2 belong to the solution, the resulting instance is $(D-\{x_1, x_2\}, k-2, r)$.

The branching vector for this case is $(1, 2, 2, 2, 2)$.

Hereafter, we assume that no two vertices of in-degree 2 have a common in-neighbour. We consider the following cases based on the degree of the in-neighbours of vertices with in-degree 2.

Branching Rule 5 If $v \in V(D)$ such that $d^-(v) = 2$ and $d^+(v) = 0$, let u_1, u_2 be the two in-neighbours of v. In this case, the algorithm considers the following sub-cases based on the out-degrees of u_1 and u_2.

Case 5.a. If at least one of u_1, u_2 has out-degree at least 2 (without loss of generality, let $d^+(u_1) \geq 2$), let x be the other out-neighbour of u_1.

Observe that $d^-(x) = 1$, as otherwise x and v are two vertices of in-degree 2 and u_1 is a common in-neighbour of x and v, and hence, case 4 would be applicable. Also $x \neq r$ because $d^-(r) = 0$ otherwise reduction rule 8.2.5 would be applicable. In this case, the algorithm branches as follows.

- When u_1 belongs to the solution, then delete u_1 from the graph. In the resulting graph $d^-(x) = 0$ and hence reduction rule 8.2.4 is applicable. Thus the resulting instance is $(D - \{u_1, x\}, k-2, r)$.
- When u_1 does not belong to the solution, then either u_2 belongs to the solution or v belongs to the solution. Therefore, the algorithm branches as follows. In

the first branch the resulting instance is $(D - \{u_2\}, k-1, r)$ and in the second branch the resulting instance is $(D - \{v\}, k-1, r)$.

The branching vector for this case is $(2, 1, 1)$.

Case 5.b. Otherwise, both u_1 and u_2 have out-degree 1. In this case, we first prove that if there is an out-tree deletion set in D, say S, of size at most k such that $v \in S$ and $u_1, u_2 \notin S$, then $S' = (S \setminus \{v\}) \cup \{u_1\}$ is also an out-tree deletion set in D. Let $F = D - S$. Note that u_1 and u_2 are leaves in the out-tree F. Let F' be the out-tree obtained from F after deleting u_1 and adding the vertex v and the edge (u_2, v). Clearly F' is an out-tree and $F' = D - S'$. Therefore, S' is an out-tree deletion set of D.

Thus, it is enough to branch as follows.

- When u_1 belongs to the solution, the resulting instance is $(D - \{u_1\}, k-1, r)$.
- When u_2 belongs to the solution, the resulting instance is $(D - \{u_2\}, k-1, r)$.

The branching vector for this case is $(1, 1)$.

In the upcoming lemma, we show that when all the reduction rules and branching rules have been considered exhaustively, all the vertices of the resulting instance, except r, have in-degree exactly 1 and in-degree of r is 0. In the later lemma we show that such an instance can be solved in polynomial time. Thus, after the exhaustive application of the reduction rules and the above mentioned branching rules, the algorithm uses the procedure described in Lemma 8.2.2 to solve the instance.

Lemma 8.2.1. *Let (D, k, r) be the instance where none of the above-mentioned reduction rules or branching rules are applicable. Then, for all $v \in V(D) \setminus \{r\}$, $d^-(v) = 1$ and $d^-(r) = 0$.*

Proof. Since reduction rule 8.2.5 is no longer applicable, we have $d^-(r) = 0$. Also since reduction rules 8.2.4 is not applicable, $d^-(v) > 0$, for all $v \in V(D) \setminus \{r\}$. To show that for any vertex $v \in V(D) \setminus \{r\}$, $d^-(v) = 1$ we proceed as follows.

Let v be some vertex in $V(D) \setminus \{r\}$. For the sake of contradiction, let $d^-(v) > 1$. If $d^-(v) > 1$ and $r \in N^-(v)$, then branching rule 1 would be applicable. Thus, we can safely assume that $r \notin N^-(v)$.

We split the situation that $d^-(v) > 1$, into the following cases.

- If $d^-(v) \geq 3$, then branching rule 2 would be applicable.
- Otherwise $1 < d^-(v) \leq 2$, that is, $d^-(v) = 2$. We split this situation into the following 3 exhaustive cases.
  - If $d^+(v) \geq 1$, then branching rule 3 would be applicable.
  - Otherwise, $d^+(v) = 0$. In this situation, if the graph has some 2 vertices of in-degree 2 with a common in-neighbour then branching rule 4 is applicable, otherwise branching rule 5 would be applicable.

Since, none of the reduction rules or branching rules are applicable, we conclude that for all $v \in V(D) \setminus \{r\}$, $d^-(v) = 1$. □

Lemma 8.2.2. *Let (D, k, r) be an instance of* Restricted Pumpkin Vertex Deletion Set *such that for all $v \in V(D) \setminus \{r\}$, $d^-(v) = 1$, and $d^-(r) = 0$. Then the instance (D, k, s, t) can be solved in $\mathcal{O}(n^{\mathcal{O}(1)})$ time, where $n = |V(D)|$.*

Proof. Since, for all $v \in V(D) \setminus \{r\}$, $d^-(v) = 1$ and $d^-(r) = 0$, and observe that (D, k, r) is a Yes instance if and only if D is an out-tree with r as the root vertex and $k \geq 0$. □

In the next lemma, we formally prove the correctness of our algorithm.

Lemma 8.2.3. *The algorithm for* Restricted Out-Tree Vertex Deletion Set *described above is correct.*

Proof. Let $I = (D, k, r)$ be an instance of ROTVDS. We prove the correctness of the algorithm by induction on $\mu = \mu(I) = k$. The base case occurs in one of the following cases.

- If $\mu \leq 0$, the algorithm correctly concludes whether (D, k, r) is a Yes / No instance from reduction rules 8.2.1 to 8.2.3.
- If $\mu \geq 0$ and D is an out-tree, the algorithm correctly concludes that (D, k, s, t) is a Yes instance from reduction rule 8.2.3.
- If for all $v \in V(D) \setminus \{r\}$, $d^-(v) = 1$ and $d^-(r) = 0$, then from Lemma 8.2.2, (D, k, r) is a Yes instance if and only if $k \geq 0$ and D is an out-tree with r as its root vertex.

Branching Rule (BR)	Case/Sub-Case	Branch Vector	c^{μ}
BR 1		$(1,1)$	2^{μ}
BR 2		$(1,2,2,2)$	2.303^{μ}
BR 3	a.i a.ii b	$(1,1,2)$ $(1,2,2,2,2)$ $(2,1,1)$	2.41422^{μ} 2.562^{μ} 2.41422^{μ}
BR 4	a b	$(1,1,2)$ $(1,2,2,2,2)$	2.41422^{μ} 2.562^{μ}
BR 5	a b	$(2,1,1)$ $(1,1)$	2.41422^{μ} 2^{μ}

Table 8.3: The branch vectors and their corresponding running times for ROTVDS.

- If r does not belong to $V(D)$, the algorithm correctly concludes that (D, k, r) is a NO instance from reduction rule 8.2.7.

By induction hypothesis we assume that for all $\mu \leq l$, the algorithm is correct. We will now prove that the algorithm is correct when $\mu = l + 1$. The algorithm does one of the following. Either applies one of the reduction rules if applicable. By the safeness of the reduction rules we either correctly conclude that I is a YES/ NO instance or produces an equivalent instance I' with $\mu(I') \leq \mu(I)$. If $\mu(I') < \mu(I)$, then by induction hypothesis and safeness of the reduction rules the algorithm correctly decides if I is a YES instance or not. Otherwise, $\mu(I') = \mu(I)$. If none of the reduction rules are applicable then the algorithm applies the first applicable Branching Rules. If some branching rule is applicable then, since μ decreases in each of the branch by at least one, by induction hypothesis the algorithm correctly concludes that I is a YES/ NO instance. If none of the branching rules are applicable, then from Lemma 8.2.2, if $I = (D, k, s, t)$, then for all $v \in V(D) \setminus \{r\}$, $d^-(v) = 1$ and $d^-(r) = 0$. Thus, in this case the base case appears and hence, we conclude that the algorithm always outputs the correct answer. □

Next, we analyse the running of the algorithm presented.

Theorem 8.2.1. *The algorithm described above solves* RESTRICTED OUT-TREE VERTEX DELETION SET *in time* $\mathcal{O}^{\star}((1+\sqrt{3})^k)$.

Proof. The reduction rules 8.2.1 to 8.2.7 can be applied in time polynomial in the input size. Also, at each of the branch we spend a polynomial amount of time. At each of the recursive calls in a branch, the measure μ decreases at least by 1. When $\mu \leq 0$, then we are able to solve the remaining instance in polynomial time

or correctly conclude that the corresponding branch cannot lead to a solution. At the start of the algorithm $\mu = k$.

The worst-case branching vector for the algorithm is $(1, 2, 2, 2, 2)$ (see Table 8.3). The recurrence for the worst case branching vector is:

$$T(\mu) \leq T(\mu - 1) + 4T(\mu - 2)$$

The running time corresponding to the above recurrence relation is $\mathcal{O}^\star(2.562^k)$. □

Theorem 8.2.1 together with the guessing step of r described in the beginning of the section gives us the following theorem.

Theorem 8.2.2. Out-Tree Vertex Deletion Set *can be solved in time* $\mathcal{O}^\star((1+\sqrt{3})^k)$.

8.3 FPT Algorithm for Out-Forest Vertex Deletion Set

In this section, we give a branching algorithm for Out-Forest Vertex Deletion Set (OFVDS). We start with a small description of our algorithm. Let (D, k) be an instance of OTVDS. Unlike our algorithm for PVDS or OTVDS, this algorithm does not require the initial guessing step. Also, the reduction rules and branching rules are designed so that when they are not applicable, the graph is empty. In other words, at any point of time, there always exists a vertex for which some reduction rule or branching rule is applicable. To achieve this trivial base case, the algorithm first eliminates (by branching over them) all vertices with in-degree at least 3, followed by the elimination of vertices with in-degree 0, then in-degree 1 vertices and finally vertices with in-degree 2. In the intermediate cases, it branches on vertices based on other factors like their out-degree and common-neighbors. These intermediate cases help the algorithm to branch in the main cases efficiently.

Next, we proceed to the details of the algorithm. The measure μ that is used to bound the depth of the search tree is the solution size, that is $\mu(D, k) = k$.

With a slight abuse of notation, in the following, during the application of any reduction/branching rule we will refer to (D, k, s, t) as the instance that is reduced with respect to the rules in higher preference order. The algorithm first applies the following reduction rules exhaustively.

Reduction Rule 8.3.1. *If $k < 0$, or $k = 0$ and D is not an out-forest, then return (D, k) is a* NO *instance.*

Reduction Rule 8.3.2. *If $k \geq 0$ and D is an out-forest, then return (D, k) is a* YES *instance.*

The safeness of reduction rule 8.3.1 and 8.3.2 is easy to see. Next we give some of the reduction rules which will eliminate certain irrelevant vertices in the digraph.

Reduction Rule 8.3.3. *For any $v \in V(D)$, if $d^+(v) = 0$ and $d^-(v) \leq 1$ then delete v. That is, the resulting instance is $(D - \{v\}, k)$.*

Reduction Rule 8.3.4. *Let $(u, v) \in E(D)$ such that $d^-(v) = 1$, $d^+(u) = 1$ and all the out-neighbours of v have in-degree exactly 1. Let $\{w_1, \ldots, w_l\}$ be the out-neighbours of v. Delete v from the graph and add the edges $\{(u, w_i) \mid i \in [l]\}$ to the graph. Let D' be the resulting graph. That is, the resulting instance is (D', k).*

Reduction Rule 8.3.5. *Let $v \in V(D)$ such that $d^-(v) = 0$ and all out-neighbours of v have in-degree exactly 1. Then delete v from the graph. That is, the resulting instance is $(D - \{v\}, k)$.*

Observe that each of the above-mentioned reduction rules can be applied in polynomial time. The safeness of reduction rules 8.3.3 to 8.3.5 is given by Lemma 8.3.1 to Lemma 8.3.3.

Lemma 8.3.1. *Reduction rule 8.3.3 is safe.*

Proof. Let $v \in V(D)$ be a vertex such that $d^+(v) = 0$ and $d^-(v) \leq 1$. Let $D' = D - \{v\}$. We need to prove that (D, k) is a YES instance if and only if (D', k) is a YES instance. For the forward direction, let S be an out-forest deletion set in D of size at most k. Since $D' - (S \setminus \{v\})$ is a sub-graph of $D - S$, therefore, it follows that $S \setminus \{v\}$ is an out-forest deletion set in D'.

For the backward direction, let S be an out-forest deletion set in D' of size at most k. If $d^-(v) = 0$, then clearly, S is also an out-forest deletion set in D. Otherwise, let u be the unique in-neighbour of v in D. If $u \in S$, then $(D' - S) \cup \{v\}$ is an out-forest with v as an isolated vertex. Otherwise, $u \notin S$, then $(D' - S) \cup \{v\}$ is an out-forest where v is in the out-tree of $D' - S$ that contains u. Therefore, S is an out-forest deletion set in D. □

Lemma 8.3.2. *Reduction rule 8.3.4 is safe.*

Proof. Let $(u, v) \in E(D)$ such that $d^-(v) = 1$ and all the out-neighbours of v have in-degree exactly 1. Let $\{w_1, \ldots, w_l\}$ be the out-neighbours of v. Also, D' be the graph obtained after deleting v from D and adding the edges $\{(u, w_i) \mid i \in [l]\}$ to D. We need to prove that (D, k) is a YES instance if and only if (D', k) is a YES instance. For the forward direction, let S be an out-forest deletion set in D of size at most k, $F = D - S$ and $W = \{w_1, \ldots, w_l\} \setminus S$. If $u \in S$, then $F - \{v\} = D' - S$. Thus, S is an out-forest of D'. If $u \notin S$ and $v \in S$, then let $S' = (S \setminus \{v\}) \cup \{u\}$. Then $F - \{u\} = D' - S'$. Hence, S' is an out-forest deletion set in D' of size at most k. Otherwise, $u, v \notin S$. Let F' be a digraph where $V(F') = V(F) \setminus \{v\}$ and $E(F') = (E(F) \setminus (u, v)) \cup \{(u, w_i) \mid w_i \in W\}$. Observe that F' is obtained after contracting the edge (u, v) and out-forests are closed under contraction. Therefore, F' is an out-forest and $F' = D' - S$. Hence S is an out-forest deletion set of D'.

For the backward direction, let S be an out-forest deletion set in D' of size at most k, $F = D' - S$ and $W = \{w_1, \ldots, w_l\} \setminus S$. Suppose $u \notin S$. Let F' be a digraph where $V(F') = V(F) \cup \{v\}$ and $E(F') = (E(F) \setminus \{(u, w_i) \mid w_i \in W\}) \cup \{(v, w_i) \mid w_i \in W\} \cup (u, v)$. Clearly F' is an out-forest and $F' = D - S$. Therefore, S is an out-forest deletion set in D. On the other hand, if $u \in S$ then, since each $w_i \in W$ has in-degree exactly 1 in D', each $w_i \in W$ is a root in F. Let F' be a digraph where $V(F') = V(F) \cup \{v\}$ and $E(F') = E(F) \cup \{(v, w_i) \mid w_i \in W\}$. Clearly F' is an out-forest and $F' = D - S$. Hence S is an out-forest deletion set of D. □

Lemma 8.3.3. *Reduction rule 8.3.5 is safe.*

Proof. Let $v \in V(D)$ such that $d^-(v) = 0$, $w_1, \ldots, w_l$ are the out-neighbours of v such that for all $i \in [l]$, $d^-(w_i) = 1$. Let $D' = D - v$. We need to prove that (D, k) is a YES instance if and only if (D', k) is a YES instance. For the forward direction, let S be an out-forest deletion set in D of size at most k. Then $S \setminus \{v\}$ is an out-forest deletion set of $D - v$. For the backward direction, let S be an out-forest deletion set in D' of size at most k, $F = D - S$ and $W = \{w_1, \ldots, w_l\} \setminus S$. Note that for all $i \in [l]$, w_i has in-degree 0 in D'. For all $w_i \in W$, let T_i be the out-tree of F containing w_i. Note that there is a unique T_i for each w_i and w_i is the root of T_i. Consider another out-tree T where $V(T) = \cup_{w_i \in W} V(T_i) \cup \{v\}$ and $E(T_i) = \cup_{w_i \in W} V(T_i) \cup \{(v, w_i) \mid w_i \in W\}$. Clearly $(F - \cup_{w_i \in W} V(T_i)) \cup T$ is an out-forest of $D - S$. Hence S is an out-forest deletion set of D. □

We now describe the branching rules used by the algorithm. Our algorithm applies 5 branching rules in order. Before giving the details of the branching rules,

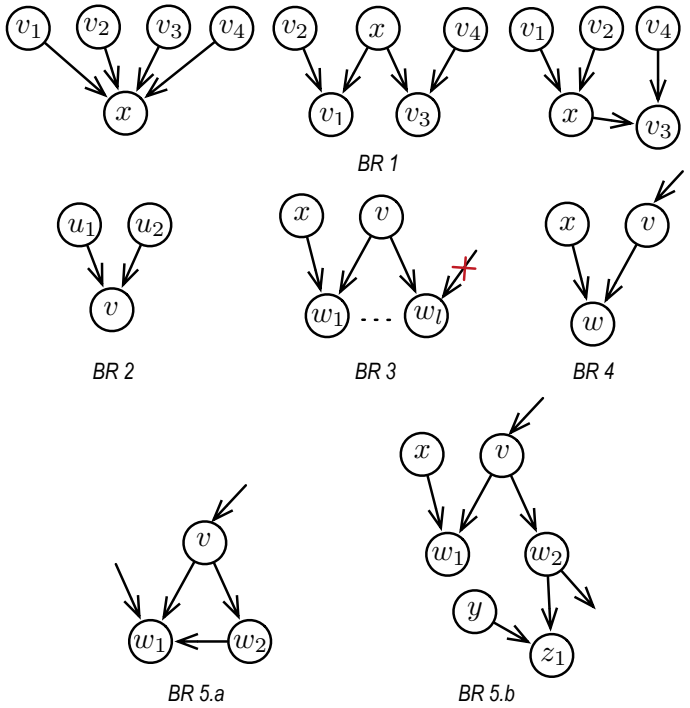


Figure 8.3: The structures in the branching rules of OFVDS

we first mention the invariants maintained by the algorithm after the exhaustive application of each of the branching rules.

Branching Rule 1 For all $v \in V(D)$, $d^-(v) \leq 2$. Also, if for any $v_1, v_2 \in V(D)$ such that $d^-(v_1) = d^+(v_2) = 2$, then v_1 and v_2 have no common in-neighbour.

Branching Rule 2 For all $v \in V(D)$, $d^+(v) \geq 1$.

Branching Rule 3 For all $v \in V(D)$, $d^-(v) \geq 1$.

Branching Rule 4 For all $v \in V(D)$, if $d^-(v) = 1$, then $d^+(v) \geq 2$.

Branching Rule 5 D is an empty graph.

Since after the exhaustive application of branching rules $1-5$, the graph is empty, either reduction rule 8.3.1 or 8.3.2 would be applicable. In other words, when none of the branching rules are applicable, the input instance will be trivial to solve.

We now give the description of the branching rules. From the description of the branching rules it is easy to see that each branching rule is exhaustive.

We first define a *hitting triplet*. For any $v_1, v_2, v_3 \in V(D)$, (v_1, v_2, v_3) is called a *hitting triplet* in D if there exists $i \in [3]$ such that v_i has in-degree at least 2 and

v_j, v_ℓ, $j, \ell \in [3] \setminus \{i\}$, are some in-neighbours of v_i. Observe that if (v_1, v_2, v_3) is a hitting triplet in D, then any out-forest deletion set of D contains at least one of v_1, v_2 or v_3.

Branching Rule 1 If (v_1, v_2, x) and (v_3, v_4, x) are two distinct hitting triplets in D then branch as follows.

- When x belongs to the solution, the resulting instance is $(D - \{x\}, k - 1)$.
- When x does not belong to the solution, at least one of v_1, v_2 and one of v_3, v_4 belongs to the solution. When v_1, v_2, v_3, v_4 are distinct, the resulting instances in the respective branches are as follows.
  - When v_1, v_3 belongs to the solution, the resulting instance is $(D-\{v_1, v_3\}, k-2)$.
  - When v_1, v_4 belongs to the solution, the resulting instance is $(D-\{v_1, v_4\}, k-2)$.
  - When v_2, v_3 belongs to the solution, the resulting instance is $(D-\{v_2, v_3\}, k-2)$.
  - When v_2, v_4 belongs to the solution, the resulting instance is $(D-\{v_2, v_4\}, k-2)$.

  We are now in the case when v_1, v_2, v_3, v_4 are not all distinct. Since (v_1, v_2, x) and (v_3, v_4, x) are hitting triplets, $v_1 \neq v_2$ and $v_3 \neq v_4$. Without loss of generality, let $v_1 = v_3$ (the other cases are symmetric). Since (v_1, v_2, x) and (v_3, v_4, x) are distinct hitting triplets and $v_1 = v_3$, $v_2 \neq v_4$. In this case, the resulting instances in the respective branches are as follows.
  - When v_1 belongs to the solution, the resulting instance is $(D-\{v_1\}, k-1)$.
  - When v_1 does not belong to the solution (and x does not belong to the solution), both v_2 and v_4 belong to the solution. In this case, the resulting instance is $(D - \{v_2, v_4\}, k - 2)$.

The branching vector for this rule is either $(1, 2, 2, 2, 2)$ or $(1, 1, 2)$ depending on the case we are in.

Observe that after the application of Branching Rule 1, for each $v \in V(D)$, $d^-(v) \leq 2$. Suppose not. Let x, y, z be some three in-neighbours of v, then (v, x, y)

and (v, x, z) form hitting triplets in D and hence Branching Rule 1 is applicable. Also, after the application of this branching rule, for any $v_1, v_2 \in V(D)$ such that $d^-(v_1), d^-(v_2) = 2$, v_1 and v_2 have no common in-neighbour.

Branching Rule 2 If $v \in V(D)$ is such that $d^+(v) = 0$, then since reduction rule 8.3.3 and branching rule 1 are not applicable, $d^-(v) = 2$.

Let u_1, u_2 be the in-neighbours of v. Note that one of v, u_1, u_2 must belong to the solution. Now observe that if there is an out-forest deletion set of D, say S, such that $v \in S$ then $(S \setminus \{v\}) \cup \{u_i\}$, for any $i \in [2]$, is also an out-forest deletion set of D. Thus in this case it is enough to branch as follows.

- When u_1 belongs to the solution, the resulting instance is $(D - \{u_1\}, k - 1)$.
- When u_2 belongs to the solution, the resulting instance is $(D - \{u_2\}, k - 1)$.

The branching vector for this rule is $(1, 1)$.

The next rule handles vertices that have in-degree 0. Note that if there is a vertex with in-degree 0 and its out-degree is 0 then reduction rule 8.3.3 would be applicable. If its in-degree is 0 and all its out-neighbours have in-degree exactly 1 then reduction rule 8.3.5 would be applicable. If two of its out-neighbours have in-degree 2 then Branching Rule 1 would be applicable. Thus, if none of the reduction rules and the above-mentioned branching rules are applicable, then any vertex which has in-degree 0 has at least one out-neighbour and exactly one of its out-neighbours has in-degree exactly two.

Branching Rule 3 If $v \in V(D)$ such that $d^-(v) = 0$, $\{w_1, \ldots, w_l\}$ are the out-neighbours of v ($l \geq 1$), for all $i \in \{2, \ldots, l\}$, $d^-(w_i) = 1$ and $d^-(w_1) = 2$, then let x be the other in-neighbour of w_1 (x may be one of w_i).

In this case, we claim that if S is an out-forest deletion set of D that contains v then $(S \setminus \{v\}) \cup \{w_1\}$ is also an out-forest deletion set of D. For the proof of this, observe that each $w_i \in \{w_1, \ldots, w_l\} \setminus S$ is a root in $D - S (= F$, say), except probably w_1. Also $d^-(v) = 0$. Therefore $(F \cup \{v\}) \setminus \{w_1\}$ is also an out-forest. Thus, S' is an out-forest deletion set of D. Therefore, it is enough to branch as follows.

- When x belongs to the solution, the resulting instance is $(D - \{x\}, k - 1)$.
- When w_1 belongs to the solution, the resulting instance is $(D - \{w_1\}, k - 1)$.

The branching vector in this branching rule is (1, 1). Again, note that hereafter, every vertex in the digraph has in-degree either 1 or 2. The upcoming branching rules 4 and 5, deal with vertices that have in-degree 1.

If $v \in V(D)$ such that $d^-(v) = 1$ and $d^+(v) = 1$, then if the unique out-neighbour, say w, of v has out-degree 0, then either reduction rule 8.3.3 or branching rule 2 would be applicable. If $d^-(w) = 1$ then, if $d^+(w) = 1$ then reduction rule 8.3.4 would be applicable, and if $d^+(w) \geq 2$ then reduction rule 8.3.4 would be applicable. Thus the case when in-degree of w is 1, is handled.

Branching rule 4 deals with the situation when there is a vertex v with in-degree 1 and out-degree 1 and its unique out-neighbour has in-degree 2. Thus, when none of the reduction rules and branching rules 1 to 4 are applicable, for any vertex v, either $d^-(v) = 2$ and $d^+(v) \geq 1$, or if $d^-(v) = 1$ then $d^+(v) \geq 2$.

Branching Rule 4 If $v \in V(D)$ such that $d^-(v) = 1$ and $d^+(v) = 1$, then if the unique out-neighbour, say w, has in-degree 2, then let x be the other in-neighbour of w.

We first make the following claim. If S is an out-forest deletion set in D such that $v \in S$, then $(S \setminus \{v\}) \cup \{w\}$ is also an out-forest deletion set in D. This is true because if $F = D - S$ is an out-forest then so is $(F - \{w\}) \cup \{v\}$. Thus, in this case, the algorithm branches on x as follows.

- When x belongs to the solution, the resulting instance is $(D - \{x\}, k - 1)$.
- When w belongs to the solution, the resulting instance is $(D - \{w\}, k - 1)$.

The branching vector for this rule is (1, 1). Recall that, when none of the reduction rules and branching rules 1 to 4 are applicable, for any vertex v, either $d^-(v) = 2$ and $d^+(v) \geq 1$, or if $d^-(v) = 1$ then $d^+(v) \geq 2$.

Let v be a vertex with in-degree 1 and out-degree at least 2. Let $\{w_1, \ldots, w_l\}$ be the out-neighbours of v. Then if all the out-neighbours of v have in-degree exactly 1, reduction rule 8.3.4 would be applicable. Also, if two of the in-neighbours have in-degree 2, then branching rule 1 would be applicable. Thus, without loss of generality assume that $d^-(w_1) = 2$ and for all $i \in \{2, \ldots, l\}$ $d^-(w_i) = 1$. Let x be the other in-neighbour of w_1. The case when $x = w_2$ will be handled in case 5.*a*. Otherwise, x, v, w_1, w_2 are all distinct. Now observe that $d^+(w_2) \geq 2$, otherwise branching rule 4 was applicable on w_2. Also, all the out-neighbours of w_2 except 1

have in-degree 1, otherwise either reduction rule 8.3.4 was applicable or branching rule 1 was applicable on w_2. Let z_1 be the out-neighbour of w_2 of in-degree 2. Let y be the other in-neighbour of z_1. Observe that (v, w_1, x) and (w_2, z_1, y) are hitting triplets in D. If (v, w_1, x) and (w_2, z_1, y) are not distinct hitting triplets, then $x = w_2$ and we are in Case 5.*a*. Otherwise, since Branching Rule 1 is not applicable, all of v, w_1, x, w_2, z_1, y are distinct. This case is handled in Case 5.*b*.

After presenting the Cases 5.*a* and 5.*b*, we will discuss why after the exhaustive application of all the described reduction and branching rules, there is no vertex v such that either $d^-(v) = 2$ and $d^+(v) \geq 1$. This will show that after the exhaustive application of all the described reduction and branching rules, the graph is empty.

Branching Rule 5 Let $v \in V(D)$ be such that $d^-(v) = 1$ and $d^+(v) = l \geq 2$. Let $\{w_1, \ldots, w_l\}$ be the set of out neighbours of v and let $d^-(w_1) = 2$ and $d^-(w_i) = 1$ for all $i \in \{2, \ldots, l\}$. Let x be the other in-neighbour of w_1. Let $d^+(w_2) = p \geq 2$. Let $\{z_1, \ldots, z_p\}$ be the out-neighbours of w_2 and let $d^-(z_1) = 2$ and for all $i \in \{2, \ldots, p\}$, $d^-(z_i) = 1$. Let y be the other in-neighbour of z_1.

In this case, the algorithm considers the following sub-cases.

Case 5.a. If $x = w_2$, then observe that $(w_2, w_1) \in E(D)$. In this case, the algorithm proceeds as follows.

We first claim the following. If S is an out-forest deletion set of D such that $w_2 \in S$, then $S' = (S \setminus \{w_2\}) \cup \{v\}$ is an out-forest deletion set of D. To prove this, let $F = D - S$. Note that each out-neighbour of w_2 in F except possibly w_1, is a root of some out-tree in F. This is because for all the out-neighbours of w_2, except w_1, w_2 is their unique in-neighbour. In fact, in $D - (S \cup \{v\})$, every neighbour of w_2 in $D - (S \cup \{v\})$ is an out-neighbour of w_2 in D and is a root of some out-tree in $D - (S \cup \{v\})$. Therefore, $D \setminus ((S \cup \{v\}) \setminus \{w_2\})$ is an out-forest. In other words, $(S \cup \{v\}) \setminus \{w_2\}$ is a solution of size at most k in D. This finishes the proof of the claim.

Thus, the algorithm branches as follows.

- When w_1 belongs to the solution, the resulting instance is $(D - \{w_1\}, k - 1)$.
- When w_1 does not belong to the solution and (D, k) is a YES instance, then there is a solution that contains v. Therefore the resulting instance is $(D - \{v\}, k - 1)$.

The branching vector in this case is $(1, 1)$.

Case 5.b. Let v, w_1, x, w_2, z_1, y be all distinct vertices. In this case, the algorithm branches as follows.

- When x belongs to the solution, the resulting instance is $(D - \{x\}, k - 1)$.
- When x does not belong to the solution, the algorithm branches on w_1 as follows.
  - When w_1 belongs to the solution, the resulting instance is $(D - \{w_1\}, k - 1)$.
  - When w_1 does not belong to the solution, then v must belong to the solution. Therefore delete v. Note that after deleting v from the graph w_2 becomes an in-degree 0 vertex and all its out-neighbours have in-degree exactly 1 except for z_1. Therefore as argued before, if there is a solution that contains w_2 then there is another solution that avoids w_2 and contains z_1. Thus, the algorithm branches as follows.
    - When y belongs to the solution, the resulting instance is $(D - \{v, y\}, k - 2)$.
    - When y does not belong to the solution, delete z_1 from the graph, that is, the resulting instance is $(D - \{v, z_1\}, k - 2)$.

The branching vector in this case is $(1, 1, 2, 2)$.

Let $v \in V(D)$ be such that $d^-(v) = 2$ and $d^+(v) \geq 1$. Let w be some out-neighbour of v. If in-degree of w is 1, then reduction rule 8.3.3, branching rule 4 or branching rule 5 would have applied on w. Otherwise, in-degree of w is 2. Let x be the other in-neighbour of w and u_1, u_2 be the in-neighbours of v. In this case, observe that (x, w, v) and (v, u_1, u_2) are hitting triplets in D and hence Branching Rule 1 is applicable.

Observe that when none of the above-mentioned reduction rules and cases are applicable, the graph is empty, that is there are no-vertices in the graph.

The following theorem proves the correctness of the algorithm presented.

Theorem 8.3.1. *The presented algorithm for* Out-Forest Vertex Deletion Set *is correct.*

Branching Rule (BR)	Case/Sub-Case	Branch Vector	c^{μ}
BR 1		$(1,2,2,2)$ $(1,1,2)$	2.303^{μ} 2.41422^{μ}
BR 2		$(1,1)$	2^{μ}
BR 3		$(1,1)$	2^{μ}
BR 4		$(1,1)$	2^{μ}
BR 5	a b	$(1,1)$ $(1,1,2,2)$	2^{μ} 2.7321^{μ}

Table 8.4: The branch vectors and their corresponding running times for OFVDS.

Proof. Let $I = (D, k)$ be an instance of Out-Forest Vertex Deletion Set. We prove the correctness of the algorithm by induction on $\mu = \mu(I) = k$. The base case occurs in one of the following cases.

- $\mu <= 0$ we correctly conclude whether (D, k) is a yes instance or not by reduction rule 8.3.1 or reduction rule 8.3.2.
- $\mu > 0$ and D is an out-forest then we correctly conclude that (D, k) is a Yes instance by reduction rule 8.3.2.

By induction hypothesis we assume that for all $\mu \leq l$, the algorithm is correct. We will now prove that the algorithm is correct when $\mu = l + 1$. The algorithm does the following. It applies one of the reduction rules if applicable. By the safeness of the reduction rules the algorithm either correctly concludes that I is a Yes/ No instance or produce an equivalent instance I' with $\mu(I') \leq \mu(I)$. If $\mu(I') < \mu(I)$, then by induction hypothesis and safeness of the reduction rules the algorithm correctly decides if I is a yes instance or not. Otherwise, $\mu(I') = \mu(I)$. If none of the reduction rules are applicable then the algorithm applies the first applicable Branching Rules. Branching Rules are exhaustive and covers all possible cases. Furthermore, μ decreases in each of the branch by at least one. Therefore, by the induction hypothesis, the algorithm correctly decides whether I is a yes instance or not. □

Theorem 8.3.2. Out-Forest Vertex Deletion Set *can be solved in* $\mathcal{O}^*((1+\sqrt{3})^k)$.

Proof. The reduction rules 8.3.1 to 8.3.5 can be applied in time polynomial in the input size. Also, at each of the branch we spend a polynomial amount of time. At each of the recursive calls in a branch, the measure μ decreases by at least 1.

When $\mu \leq 0$, then reduction rule 8.3.1 or 8.3.2 is applicable and hence the algorithm correctly returns the answer and terminate. At the start of the algorithm $\mu = k$.

The worst-case branching vector for the algorithm is $(1, 1, 2, 2)$ (see Table 8.4). The recurrence for the worst case branching vector is:

$$T(\mu) \leq 2T(\mu - 1) + 2T(\mu - 2)$$

The running time corresponding to the above recurrence relation is $\mathcal{O}^{\star}((1 + \sqrt{3})^k)$. □

Chapter 9

Kernel for Deletion to Bounded Treewidth DAGs

Recall the objective of the paper of Mnich and van Leeuwen [164] from Chapter 6. To make progress on the question of the existence of a polynomial kernel for DFVS, Mnich and van Leeuwen [164] considered DFVS with an additional restriction on *the output rather than the input*. Specifically, the output digraphs considered where out-forest, out-tree and pumpkin. Note that all these output digraphs are acyclic digraphs with treewidth at most 2 (throughout the chapter, by treewidth of a digraph we mean the treewidth of its underlying undirected graph.) In this chapter we, in a way, unify and extend the results of Mnich and van Leeuwen [164] by giving an outline of a kernel for the problem of deletion to DAGs of bounded treewidth. Towards doing this, we observe that the existing machinery can already be harnessed to resolve this problem affirmatively. Nevertheless, as this finding already generalizes some results in the literature, we give an outline of this result here.

We would also like to remark that the study of the DFVS problem by restricting the input digraph class (as done in Chapter 3) or by parameterizing it by a non-trivial structural parameter that is larger than solution size (as done in Chapter 5) has a major advantage over studying the DFVS problem by restricting the resulting DAG. This is because the first two restrictions do not alter the problem at hand, namely, the focus is still aimed at DFVS itself rather than at a variant of DFVS. More precisely, the later approach may actually derail us from the track of resolving the kernelization complexity of DFVS as each restriction of the output DAG results in its own definition of a variant of DFVS that may have its own properties. Thus, if the ultimate goal is to design a polynomial kernel for DFVS itself (or prove that such

a kernel does not exist), we find the first approach more suitable. Nevertheless, it is also important to note that the questions raised by restricting the output, namely, the study of the variants of DFVS, may be interesting in their own right. We now proceed to the main result in this chapter.

For a positive integer $\eta > 0$, let $\mathcal{F}_\eta$ denote the family of digraphs of treewidth at most η.

$\mathcal{F}_\eta$-Vertex Deletion	**Parameter:** k
Input: A digraph D and a non-negative integer k.	
Question: Does there exist a set of at most k vertices, say S, such that $D - S \in \mathcal{F}_\eta$?	

Theorem 9.0.1. *$\mathcal{F}_\eta$-Vertex Deletion problem admits a polynomial kernel.*

The kernelization algorithm for $\mathcal{F}_\eta$-Vertex Deletion problem follows along the lines of kernel for Planar-$\mathcal{F}$-Deletion given in [99]. We follow the steps of kernelization algorithm given for Planar-$\mathcal{F}$-Deletion in [99] to design a polynomial kernel for $\mathcal{F}_\eta$-Vertex Deletion Set.

Let D be a digraph and let S be a vertex subset such that $D - S$ is a DAG and has treewidth at most η. Then, we call such S a *DAG-treewidth η-modulator* set. Towards kernelization, we begin by showing that any Yes instance (D, k) to $\mathcal{F}_\eta$-Vertex Deletion Set has a set X of $k^{\mathcal{O}(1)}$ vertices such that every connected component C of $D - X$ is a *near-protrusion*. Recall that a r-protrusion C in a graph D is a vertex set such that $|N(C)| \leq r$ and $\mathbf{tw}(G[C]) \leq r$. The components of $D - X$ will not necessarily be protrusions, however we will prove that there is a constant r such that if (D, k) is a yes instance, then for any DAG-treewidth η-modulator set S of size at most k, $C \setminus S$ is a r-protrusion of $D - S$.

For our kernelization algorithm we also need an approximation algorithm for $\mathcal{F}_\eta$-Vertex Deletion Set. To obtain a factor c-approximation algorithm for $\mathcal{F}_\eta$-Vertex Deletion Set we do as follows. Towards computing an approximate solution for $\mathcal{F}_\eta$-Vertex Deletion Set, we run the constant factor approximation algorithm for Treewidth-η-Modulator given in [99], on the underlying undirected graph of D. Let us recall that in the Treewidth-η-Modulator problem, we are given an undirected graph G and the objective is to find a minimum sized vertex set W such that $\mathbf{tw}(G - W) \leq \eta$. Let, W be a $c' \cdot \mathsf{OPT}$ sized approximation to the Treewidth-η-Modulator problem when run on the underlying undirected

graph of D. Here, OPT is the size of a minimum sized vertex set W such that $\mathbf{tw}(D-W) \leq \eta$. Furthermore, let OPTdag be the size of a minimum sized set $W^\star$ such that $D-W^\star$ is a DAG in $\mathcal{F}_\eta$. Observe that $\mathsf{OPT} \leq \mathsf{OPTdag}$. Note that $D-W$ is in $\mathcal{F}_\eta$, though D_W may not be DAG. To make $D-W$ a DAG we now find a set of vertices such that it is a directed feedback vertex set. We compute a smallest size directed feedback vertex set, W_{dfvs}, in $D-W$ by calling the algorithm for DFVS parameterized by treewidth given in [33]. Since, an optimum solution to $\mathcal{F}_\eta$-Vertex Deletion Set is also a directed feedback vertex set we have that $|W_{\mathsf{dfvs}}| \leq \mathsf{OPTdag}$. Now observe that $W \cup W_{\mathsf{dfvs}}$ is a solution to $\mathcal{F}_\eta$-Vertex Deletion Set of size at most $(c'+1)\mathsf{OPTdag}$. Thus, this gives a factor $c = (c'+1)$ approximation algorithm for $\mathcal{F}_\eta$-Vertex Deletion Set. Let this factor c approximation algorithm for $\mathcal{F}_\eta$-Vertex Deletion Set be called `Approx-`$\mathcal{F}_\eta$`-VDS`.

The kernelization algorithm begins by running `Approx-`$\mathcal{F}_\eta$`-VDS`, a c-approximation algorithm, for $\mathcal{F}_\eta$-Vertex Deletion Set. If the solution returned by the approximation algorithm is more than ck, the kernelization algorithm returns a trivial no instance. Otherwise, let W denote the approximate solution. Based on W, and exploiting the fact that $D-X$ has treewidth at most η, we are able to construct $R \subseteq (D-W)$ on $k^{\mathcal{O}(1)}$ vertices such that: (a) for every connected component C of $D \setminus (W \cup R)$, $|N(C) \cap R| \leq 2(\eta+1)$, and (b) for every connected component C of $D \setminus (W \cup R)$, and $u, v \in N(C) \cap W$ there are at least $k+\eta+3$ vertex disjoint paths from u to v in the underlying undirected graph of D. Now we can combine these components (or rather near protrusions) to get the following partition of $V(D)$ satisfying the below stated properties. That is,

$$V(D) = W \uplus R \uplus \bigcup_{i \in \{1,\ldots,q\}} Z_i \tag{9.1}$$

The properties satisfied by the partition given in Equation 9.1 are as follows.

1. $|R| \leq k^{\mathcal{O}(1)}$.
2. for each $i \in \{1,\ldots,q\}$, $D[Z_i]$ is acyclic and $\mathbf{tw}(D[Z_i]) \leq \eta$,
3. for each $i \in \{1,\ldots,q\}$, $N(Z_i) \subseteq W \cup R$,
4. for each $i \in \{1,\ldots,q\}$, $|N(Z_i) \cap R| \leq 2(\eta+1)$, and
5. for each $i \in \{1,\ldots,q\}$, Z_i is a $\mathcal{O}(\eta)$-near protrusion.

The proof for the partition given in Equation 9.1 is similar to the decomposition

algorithm given in Lemma 5.3.1, and in fact identical to the proof of Lemma 25 of Fomin et al. [99].

The next step of the kernelization algorithm is to bound the number q of connected components of $D - (W \cup R)$ by a polynomial in k. Towards this we can proceed in a manner identical to Lemma 36 of Fomin et al. [99]: Here it is proved that, if $W \cup R$ satisfies properties 2-5 then one can mark in polynomial time a $k^{\mathcal{O}(1)}$ size subset of the connected components with the following properties.

For every connected component Z_i that is *not* marked, every subset $S \subseteq V(D)$ such that $|S| \leq k+1$ and $\mathbf{tw}(D - (S \cup \{v\})) \leq \eta$, we have that $\mathbf{tw}(D - S) \leq \eta$.

In other words, all vertices in all of the unmarked connected components are *irrelevant* with respect to the treewidth of $D - S$ for a potential solution S. In our setting, such vertices v may still be relevant because $D - (S \cup \{v\})$ could be a DAG, while $D - S$ is not. However, this can be remedied by essentially the same strategy: For each pair u, v of vertices in $W \cup R$ (including the pairs where $u = v$), mark at most $k+10$ connected components of $D - (W \cup R)$ that contain a directed path from an out-neighbor of u to an in-neighbor of v. This results in at most $k^{\mathcal{O}(1)}$ additional components being marked. A proof following the lines of the proof of Lemma 36 of Fomin et al. [99] (but simpler, because directed paths are easier than rooted minors) shows every vertex v in an unmarked component is irrelevant in the following sense: For every subset $S \subseteq V(D)$ such that $|S| \leq k+1$ and $D - S \cup \{v\} \in \mathcal{F}_\eta$ we have that $D - S \in \mathcal{F}_\eta$. This leads to a reduction rule that, provided the number of components of $D - (W \cup R)$ is more than $k^{\mathcal{O}(1)}$, selects any vertex v in an unmarked component and deletes it. After this rule is exhaustively applied we can also conclude that the number q of connected components of $D - (W \cup R)$ is at most $k^{\mathcal{O}(1)}$.

Having bounded the number of connected components of $D - (W \cup R)$, next we need to bound their size. However, instead of bounding the size of the connected components we instead bound the *bitsize* of a representation of the component. This leads to a compression rather than a kernel, but because the compression is into a problem in NP, we can use NP-completeness to reduce back to our problem and obtain a polynomial kernel.

For each component Z_i, define its boundary $B_i = N(Z_i) \setminus S$. We know that $|B_i| = \beta_i = \mathcal{O}(\eta)$ for any solution S of size at most k. Define D_i to be a boundaried graph $D[N[Z_i] \setminus S]$ with boundary $|N(Z_i) \setminus S|$. Since the class $\mathcal{F}_\eta$ has finite state (see e.g. Bodlaender et al. [24]) the canonical equivalence relation for $\mathcal{F}_\eta$ restricted to the set of β_i-boundaried graphs has a finite set of representatives. Let D_i' be

the representative for D_i. We have that $(D - Z_i \cup S) \oplus D'_i$ is in $\mathcal{F}_\eta$ if and only if $D - S \in \mathcal{F}_\eta$.

We may think of the selection of D'_i given S as a function of B_i and $S_i = S \cap Z_i$, and write $D'_i(B_i, S_i)$ for this function. Similarly, $B_i(S) = N(Z_i) \setminus S$ and $S_i(S) = S \cap Z_i$ can be thought of as functions of S. The equivalence above shows that, for any two sets S^1 and S^2 of size at most k, if $S_1 \cap (W \cup R) = S_2 \cap (W \cup R)$ and for every i, $D'_i(B_i(S^1), S_i(S^1)) = D'_i(B_i(S^2), S_i(S^2))$ then $D - S_1 \in \mathcal{F}_\eta$ if and only if $D - S_2 \in \mathcal{F}_\eta$.

Thus, for every connected component Z_i it is sufficient to store for every $B_i \subseteq N(Z_i)$ of size $\mathcal{O}(\eta)$, for every representative D'_i of the canonical equivalence relation for $\mathcal{F}_\eta$ restricted to the set of $\mathcal{O}(\eta)$-boundaried graphs, the size of the smallest set S_i such that $D'_i(B_i, S_i) = D'_i$ (or ∞ if no such set exists or is larger than k). Hence, for each of the connected components Z_i we store a number between 0 and $k+1$ for each of the $k^{\mathcal{O}(1)}$ choices of B_i and each of the $\mathcal{O}(1)$ choices of D'_i.

Chapter 10

Sub-Exponential FPT Algorithms for DFAS, Directed Cutwidth and Optimal Linear Arrangement in Digraphs of Bounded Independence Number

In this chapter, we design sub-exponential FPT algorithms for various cut problems on the class of digraphs of bounded independence number. For a simple digraph D (every pair of vertices has at most one arc), denote $n = |V(D)|$ and $m = |E(D)|$. Let us formally define the class of digraphs relevant to our work in this chapter. Given a digraph D, a vertex subset $I \subseteq V(D)$ is called an *independent set* if there are no arcs between any pair of vertices in I. For any positive integer α, let

$$\mathcal{D}_\alpha = \{D \mid \text{ maximum independent set in } D \text{ has size at most } \alpha\}.$$

Observe that for $\alpha = 1$, $\mathcal{D}_\alpha$ is a family of tournaments. *For simplicity, we assume to work with simple digraphs. However, all our results hold also when the digraph is not simple. That is, for any pair of vertices u, v, both the arcs (u, v) and (v, u) can be present in the digraph.*

Our first theorem gives a sub-exponential time algorithm for DFAS on $\mathcal{D}_\alpha$.

Theorem 10.0.1. DFAS *on $\mathcal{D}_\alpha$ is solvable in time $2^{\mathcal{O}(\alpha^2\sqrt{k}\log(\alpha k))} \cdot n^{\mathcal{O}(\alpha)}$.*

Towards the definition of the second problem, let D be a digraph. For $X, Y \subseteq V(D)$, let $E(X,Y) = \{(u,v) \in E(D) \mid u \in X, v \in Y\}$ denote the set of arcs from X to Y. For an integer q, denote $[q] = \{1, \ldots, q\}$. The *width* of an ordering $(v_1, \ldots, v_n)$ of $V(D)$ is $\max_{i \in [n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})|$. The *cutwidth* of D, denoted by $\mathbf{ctw}(D)$, is the smallest possible width of an ordering of $V(D)$. Now, the second problem is defined as follows.

Directed Cutwidth	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Is $\mathbf{ctw}(D) \leq k$?	

We present a sub-exponential time algorithm for Directed Cutwidth on $\mathcal{D}_\alpha$.

Theorem 10.0.2. Directed Cutwidth *on* $\mathcal{D}_\alpha$ *is solvable in time* $2^{\mathcal{O}(\alpha^2\sqrt{k}\log(\alpha k))} \cdot n^{\mathcal{O}(\alpha)}$.

Towards the definition of the third problem, let D be a digraph. For two integers i, j, let $[i > j]$ evaluate to 1 if $i > j$, and to 0 otherwise. The *cost* of an ordering $\sigma = (v_1, \ldots, v_n)$ of $V(D)$ is $\sum_{(v_i,v_j) \in E(D)} (i-j) \cdot [i > j]$. In other words, every arc (v_i, v_j) directed backward in σ costs a value equal to its length, where the length of (v_i, v_j) is the distance between v_i and v_j in σ. Our last problem seeks an ordering of cost at most k.

Optimal Linear Arrangement (OLA)	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Is there an ordering of $V(D)$ of cost at most k?	

Our third theorem gives a sub-exponential time algorithm for OLA on $\mathcal{D}_\alpha$.

Theorem 10.0.3. OLA *on* $\mathcal{D}_\alpha$ *is solvable in time* $2^{\mathcal{O}(\alpha^2 k^{\frac{1}{3}}\log(\alpha k))} \cdot n^{\mathcal{O}(\alpha)}$.

10.1 Our Methods and k-Cuts

Our algorithms are based on the general framework of Fomin and Pilipczuk [102] to design parameterized sub-exponential time algorithms. The main ingredient to prove in order to employ this framework is a combinatorial upper bound on the number of "k-cuts" in graphs that are Yes instances of the problem at hand. The

proof for the combinatorial bound in our case is completely different from the proof given by Fomin and Pilipczuk [102] for transitive tournaments. The bound of Fomin and Pilipczuk [102] is achieved by mapping the set of k-cuts in a transitive tournament to the set of partitions of the integer k. Then, an asymptotic bound on the partition number of an integer yields a bound on the number of k-cuts in a transitive tournament. In the case of digraphs with bounded independence number, we do not know how to attain the desired bound by utilizing such partitions of integers.

Before we go further, we define the notion of k-cuts.

Definition 10.1.1 (k-cut)**.** *A* k-cut *in a digraph D is a partition of $V(D)$ into two parts L and R (that is, $V(D) = L \uplus R$) such that $|E(R, L)| \leq k$. The k-cut is denoted by the ordered pair (L, R). The set L is called the* left *part of the cut, and the set R is called the* right *part of the cut. The arcs in $E(R, L)$ are the* cut-arcs *of (L, R).*

Our first technical contribution is an upper bound on the number of k-cuts in $\mathcal{D}_\alpha$.

Lemma 10.1.1. *If $D \in \mathcal{D}_\alpha$, then for any positive integer k, the number of k-cuts in D is at most $2^{c\sqrt{k}\log k} \cdot (n+1)^{2\alpha\lceil\sqrt{k}\rceil} \cdot \log n$, where c is a fixed absolute constant.*

The upper bound in Lemma 10.1.1 is of the form $n^{\mathcal{O}(f(\alpha)\sqrt{k})}$. That is, it shows that the number of k-cuts in digraphs in $\mathcal{D}_\alpha$ is upper bounded by a sub-exponential function in n. Clearly, such a bound is not sufficient to design sub-exponential time parameterized algorithms. If any of the problems DFAS, Directed Cutwidth or OLA on $\mathcal{D}_\alpha$ admits a polynomial kernel, then Lemma 10.1.1 can readily yield a sub-exponential time parameterized algorithm for the corresponding problem. However, we do not know whether these problems admit such kernels, and the resolution of these questions remains an interesting open problem.

Our second main technical contribution is an upper bound on the number of k-cuts in a *subfamily* of $\mathcal{D}_\alpha$. This bound suffices to prove Theorems 10.0.1, 10.0.2 and 10.0.3 by embedding it in the framework of Fomin and Pilipczuk [102]. Let us first define this subfamily. Given a vertex $v \in V(D)$, denote the set of out-neighbors of v in D by $N^+_D(v) = \{u \in V(D) \mid (v, u) \in E(D)\}$.

Definition 10.1.2 (d-out-degenerate digraph)**.** *For any positive integer d, a digraph D is* d-out-degenerate *if for every subgraph H of D, there exists a vertex $v \in V(H)$ such that $d^+_H(v) \leq d$. An ordering $(v_1, \ldots, v_n)$ of the vertex set of D is a* d-out-degeneracy sequence *of D if for any $i \in \{2, \ldots, n\}$, $|N(v_i) \cap \{v_j \mid j < i\}| \leq d$.*

Observe that a digraph is d-out degenerate if and only if it has a d-out-degeneracy sequence, that is there is an ordering of the vertex set of the digraph such that each vertex has at most d edges to the vertices before it. Also observe that DAGs are 0-out-degenerate. Next, we define a class of digraphs having small independence number and bounded out-degeneracy. Formally, $\mathcal{D}_{\alpha,d} = \{D \mid D \in \mathcal{D}_\alpha$ and D is d-out-degenerate$\}$. Note that if $D \in \mathcal{D}_{\alpha,d}$, then every induced subgraph D' of D belongs to $\mathcal{D}_{\alpha,d}$. Our second main technical contribution is formally stated as follows.

Lemma 10.1.2. *If $D \in \mathcal{D}_{\alpha,d}$, then for any positive integer k, the number of k-cuts in D is at most $2^{c(\alpha+1)\sqrt{k}\log k} \cdot (d+1+\alpha(2k+1))^{2\alpha(\alpha+1)\lceil\sqrt{k}\rceil} \cdot \log(d+\alpha(2k+1)) \cdot n^{\alpha+1}$, where c is a fixed absolute constant.*

One can easily see that if (D, k) is a YES instance of DFAS, DIRECTED CUTWIDTH or OLA, then D is k-out-degenerate. Thus, Lemma 10.1.2 implies a sub-exponential (in k) upper bound on the number of k-cuts for YES-instances of these problems. In fact, for OLA one can show that D is $2k^{2/3}$-out-degenerate, and thus obtain an improved upper bound on the number of k-cuts for YES instances. Since the k-cuts of any digraph can be enumerated with polynomial delay [102], hence the upper bounds in Lemmas 10.1.1 and 10.1.2 are constructive.

10.2 Proof Outline for Our Results

In what follows, we present our proof strategies for the results stated above.

Proof Strategy of Lemma 10.1.1: We first make a very simple observation, which serves as the starting point of our proof. Let $V(D) = V_1 \uplus \cdots \uplus V_\ell$ be some partition of $V(D)$. Then, the number of k-cuts in D is upper bounded by the product of the number of k-cuts in the digraph induced by each V_i. Thus, we aim to partition $V(D)$ into parts that induce "sufficiently structured" subdigraphs—we want the number of k-cuts in $D[V_i]$, for any $i \in [\ell]$, to be "easier" to upper bound than the number of k-cuts in D directly. Moreover, since our aim is to achieve a bound of $n^{o(k)}$ for the total number of k-cuts in D, we want a partition $V(D) = V_1 \uplus \cdots \uplus V_\ell$ where $\ell = o(k)$. To this end, we utilize Gallai-Milgram's Theorem (explained next) under the canvas of chromatic coding.

On the one hand, Gallai-Milgram's Theorem states that if the size of a maximum independent set in a digraph is α, then its vertex set can be partitioned into at most α parts such that the digraph induced by each of these parts has a directed

Hamiltonian path. On the other hand, chromatic coding (in its derandomized form) provides a family $\mathcal{F}$ of partitions of $V(D)$ such that *(i)* $|\mathcal{F}| = 2^{o(k)} \log n$, *(ii)* for each k-cut (L, R) in D, there exists a partition $\mathcal{P} \in \mathcal{F}$ such that all the cut arcs of (L, R) go across the parts of $\mathcal{P}$, and *(iii)* the number of parts of each partition in $\mathcal{F}$ is upper bounded by $\mathcal{O}(\sqrt{k})$. If the cut-arcs of (L, R) go across the parts of a partition $\mathcal{P}$, we say that (L, R) *respects* $\mathcal{P}$. To see how to combine these two tools, let $\mathcal{F}$ be a family provided by chromatic coding. Since the number of partitions in $\mathcal{F}$ is $2^{o(k)} \log n$, and for each k-cut (L, R) there exists a partition in $\mathcal{F}$ that it respects, it suffices to bound the number of k-cuts that respect a particular (arbitrary) partition in $\mathcal{F}$. Then, the total number of k-cuts in the digraph will be the product of the number of k-cuts that respect a partition in $\mathcal{F}$, over all partitions in $\mathcal{F}$.

Consider an arbitrary partition $\mathcal{P} \in \mathcal{F}$ (of $V(D)$). Let $\mathcal{P} = P_1 \uplus \cdots \uplus P_\ell$. Recall that $\ell = \mathcal{O}(\sqrt{k})$, and the number of k-cuts in D is at most the product of the number of k-cuts in $D[P_i]$, over all $i \in [\ell]$. Here, a crucial insight is that the number of k-cuts in D that respect $\mathcal{P}$ is at most the product of the number of 0-cuts in $D[P_i]$, over all $i \in [\ell]$. Thus, we have reduced our problem to upper bounding the number of 0-cuts in a digraph. Now, to upper bound the number of 0-cuts in $D[P_i]$ by $n^{o(k)}$, we utilize Gallai-Milgram's Theorem. Since $D[P_i] \in \mathcal{D}_\alpha$, Gallai-Milgram's Theorem implies that P_i can be partitioned into at most α parts, say $P_i = P_{i1} \uplus \ldots \uplus P_{iq}$, $q \leq \alpha$, such that for each $j \in [q]$, $D[P_{ij}]$ has a directed Hamiltonian path. Thus, we have finally reduced our problem to finding 0-cuts in digraphs that have a directed Hamiltonian path. As we will see later, the number of 0-cuts in such digraphs is linear in its number of vertices. Combining everything together, we are able to bound the number of k-cuts in D by $n^{\mathcal{O}(\alpha\sqrt{k})}$.

Proof Strategy of Lemma 10.1.2: Each vertex in a digraph D has two choices of how to participate in a cut—it can belong either to its left side or to its right side. Thus, if $|V(D)| = n$, a trivial upper bound on the total number of k-cuts in D is 2^n. Suppose that we have (somehow) reached a "situation" where most of the vertices *must* belong to only one of the sides of a k-cut. Then, the arguments to attain the 2^n bound imply that the number of k-cuts is at most 2^q, where q is the number of vertices which possibly have both choices. By the bound in Lemma 10.1.1, we can further conclude that the number of k-cuts is, in fact, at most $q^{\mathcal{O}(\alpha\sqrt{k})}$. Thus, if $q = k^{\mathcal{O}(1)}$ (that is, only $k^{\mathcal{O}(1)}$ vertices can choose a side), we get a bound of $2^{o(k)}$.

On a different note, suppose that we can identify a set of vertices in D, say V_1, such that $D[V_1]$ has at most $2^{o(k)}$ k-cuts. If V_1 is large enough, say $|V_1|$ is such that $|V(D) \setminus V_1| = k^{\mathcal{O}(1)}$, then we can bound the number of k-cuts in $D[V(D) \setminus V_1]$ by $2^{o(k)}$

(by Lemma 10.1.1). Since the number of k-cuts in D is bounded by the product of the number of k-cuts in $D[V_1]$ and the number of k-cuts in $D[V(D) \setminus V_1]$, we attain the bound of $2^{o(k)}$ on the number of k-cuts in D.

Our algorithm combines the two ideas above to obtain the desired bound. For any vertex $v \in V(D)$, we aim to bound the number of k-cuts in D where v is "forced" to belong to the left part. We exploit the position of v in a fixed d-out-degeneracy sequence of D to conclude that a large number of vertices are forced to belong to one side of these cuts. Then, building on the second idea, we inductively find a set of vertices such that the digraph induced on it has independence number strictly smaller than the independence number of D. For such a set of vertices, we can inductively assume that the number of k-cuts in the digraph induced by them is $2^{o(k)}$. Having this bound at hand, we are able to conclude the proof.

Proof Strategy of Theorems 10.0.1, 10.0.2 and 10.0.3: To obtain sub-exponential FPT algorithms for DFAS, Directed Cutwidth and OLA on $\mathcal{D}_\alpha$, we first use Lemma 10.1.2 to bound the number of k-cuts in the digraphs of the Yes instances of these problems by $2^{o(k)}$. Here, we rely on the observation that these digraphs must be k-out-degenerate. Though we do not explicitly state this, the procedures to bound the number of k-cuts in both Lemmas 10.1.1 and 10.1.2 are constructive. However, constructiveness is not necessary since a standard branching procedure can also enumerate all k-cuts in a digraph with polynomial delay [102]. To actually solve any of the three problem, we rely on dynamic programming procedures over the k-cuts in the input digraph.

The last two steps of this proof (namely, the enumeration and the dynamic programming procedures) are quite standard, based on the work by Fomin and Pilipczuk [102] to obtain sub-exponential FPT algorithms for DFAS, Directed Cutwidth and OLA on tournaments.

For a set of vertices $\{v_1, \ldots, v_n\}$, let $(v_1, \ldots, v_n)$ denote the ordering where for any $i \in [n]$, v_i is the ith vertex of the ordering.

10.3 Bounding the Number of k-Cuts for Digraphs in $\mathcal{D}_\alpha$

In this section, we prove that the number of k-cuts in any digraph on n vertices with bounded independence number is $n^{o(k)}$. In particular we prove Lemma 10.1.1.

Let us recall that a k-cut in a directed graph D is a partition of the vertex set of D into two parts, $V(D) = L \uplus R$, such that $|E(R, L)| \leq k$. Let us note that a 0-cut in a digraph D is a partition (L, R) of the vertex set $V(D)$ such that there are no arcs from R to L in D.

At the heart of the proof of Lemma 10.1.1 is a simple observation that helps us focus on parts of the digraph for which bounding the number of k-cuts is easier. This simple observation is then exploited to its fullest using two main tools - (1) the Gallai-Milgram's Theorem and (2) chromatic coding. Let us state them formally. We begin by stating this key observation, followed by formally defining both these ideas.

Lemma 10.3.1. *Let D be a digraph and $k \in \mathbb{Z}^+$. Let $V(D) = V_1 \uplus \ldots \uplus V_q$ be some partition of $V(D)$. For any $i \in [q]$, let N_i be the number of k-cuts in $D[V_i]$, then the number of k-cuts in D is at most $\prod_{i \in [q]} N_i$.*

Proof. To prove the lemma, observe that, it is enough to prove that for any k-cut (L, R) in D, there exists k-cuts (L_i, R_i), for each $i \in [q]$, in $D[V_i]$, such that $L = \cup_{i \in [q]} L_i$ and $R = \cup_{i \in [q]} R_i$. To see this, for any $i \in [q]$, let $L_i = L \cap V_i$ and $R_i = R \cap V_i$. Observe that, each (L_i, R_i) is a k-cut in $D[V_i]$, otherwise (L, R) is not a k-cut in D. □

Thus, if we can partition the vertex set of D into $o(k)$ parts such that it is "easier" to bound the number of k-cuts in each of these parts, then we are done. At a high level, we will first partition the vertex set of D using chromatic coding, and then further partition each part of this partition using Gallai-Milgram's Theorem. We will then conclude by proving that the number of k-cuts in each of the sub-parts is linear in the number of vertices. We now state the Gallai-Milgram's Theorem formally.

Proposition 10.3.1 ([109], Gallai-Milgram Theorem, 1960)**.** *For any $\alpha \in \mathbb{Z}^+$ and $D \in \mathcal{D}_\alpha$, there exists a partition of $V(D) = V_1 \uplus \ldots \uplus V_q$, such that $q \leq \alpha$ and for each $i \in [q]$, $D[V_i]$ has a directed Hamiltonian path.*

Next, we state the technique of chromatic coding in its derandomized form. To this end, we first define *universal (n, k, r)-coloring family* and then state the known results about the existence of such a families of bounded size. This result is called the chromatic coding lemma. For any graph G, a *proper vertex coloring* of G is a function $f : V(G) \to \mathbb{Z}^+$, such that for any $(u, v) \in E(G)$, $f(u) \neq f(v)$.

Definition 10.3.1 ([7], Universal (n, k, r)-Coloring Family)**.** *For integers n, k and r, a family $\mathcal{H}$ of functions from $[n]$ to $[r]$ is called a universal (n, k, r)-coloring family, if for any graph G on the vertex set $[n]$ with at most k edges, there exists an $h \in \mathcal{H}$ which is a proper vertex coloring of G.*

Observe that the above mentioned definition holds for digraphs too, where the notion of proper vertex coloring is defined on its underlying undirected graph.

Proposition 10.3.2 ([7], Chromatic Coding Lemma)**.** *For any $n, k \geq 1$, there exists a universal $(n, k, 2\lceil\sqrt{k}\rceil)$-coloring family of size at most $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot \log n$.*

A formulation of the Chromatic Coding lemma, in the way that is useful to us, can be seen in the following corollary.

Corollary 10.3.1. *For any digraph D on n vertices, and an integer k, there exists a family $\mathcal{F}$ of partitions of $V(D)$ into at most $2\sqrt{k}$ parts, such that,*

1. *for any k-cut (L, R) in D, there exists a partition $\mathcal{P} = \{P_1, \ldots, P_q\}$ in the family $\mathcal{F}$, such that for any cut-arc (u, v) of (L, R), there exists $i, j \in [q]$, $i \neq j$, such that $u \in P_i$ and $v \in P_j$, and*

2. *$|\mathcal{F}| = 2^{\mathcal{O}(\sqrt{k}\log k)} \cdot \log n$.*

Proof. Let $\mathcal{H}$ be a $(n, k, 2\lceil\sqrt{k}\rceil)$-universal coloring family from Proposition 10.3.2, of size at most $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot \log n$. We construct a family $\mathcal{F}$ of partitions of $V(D)$ from the family $\mathcal{H}$ as follows. For each $h \in \mathcal{H}$, there is a partition $\mathcal{P}_h = P_1 \uplus \cdots \uplus P_{\lceil 2\sqrt{k}\rceil}$ in $\mathcal{F}$, where for any $i \in [2\lceil\sqrt{k}\rceil]$, $P_i = h^{-1}(i)$. Here, if for a certain i, $P_i = \emptyset$, then we discard this part from the partition $\mathcal{P}_h$.

We will now show that $\mathcal{F}$ is indeed the family with the required properties. Since $|\mathcal{H}| = 2^{\mathcal{O}(\sqrt{k}\log k)} \cdot \log n$, clearly $|\mathcal{F}| = 2^{\mathcal{O}(\sqrt{k}\log k)} \cdot \log n$. Let (L, R) be some k-cut in D. Consider the digraph, say $D_{(L,R)}$, on the vertex set of D with only the cut-arcs of (L, R). Note that $|E(D_{(L,R)})| \leq k$. Thus, from the definition of $(n, k, 2\lceil\sqrt{k}\rceil)$-universal coloring family, there exists a function $h : V(D_{(L,R)}) \to [2\lceil 2\sqrt{k}\rceil]$ in $\mathcal{H}$, such that h is a proper vertex coloring of $D_{(L,R)}$. Consider the partition $\mathcal{P}_h \in \mathcal{F}$. Let $\mathcal{P}_h = P_1 \uplus \cdots \uplus P_{2\lceil\sqrt{k}\rceil}$. Since h is a proper coloring of $D_{(L,R)}$ and all the cut-arcs of (L, R) are in $D_{(L,R)}$, for any cut-arc (u, v) of (L, R), $h(u) \neq h(v)$. Thus, if $h(u) = i$ and $h(v) = j$, $i \neq j$, then $u \in P_i$ and $v \in P_j$. □

For the rest of this section, let $\mathcal{F}$ denote the family described in Corollary 10.3.1 for the digraph D and integer k. For any arc (u, v) of a digraph and a partition $\mathcal{P} = P_1 \uplus \cdots \uplus P_q$ of the vertex set of the digraph, we say that the arc (u, v) *goes across the parts of this partition* $\mathcal{P}$, if $u \in P_i$, $v \in P_j$ and $i \neq j$. For any partition $\mathcal{P}$ of the vertex set of the digraph D, we say that a k-cut (L, R) in D *respects* $\mathcal{P}$ if all the cut-arcs of (L, R) go across the parts of $\mathcal{P}$. The next lemma states that, the number of k-cuts in D is at most the sum of the number of k-cuts that respect a partition $\mathcal{P}$, over all partitions $\mathcal{P} \in \mathcal{F}$. Since $|\mathcal{F}| = 2^{o(k)}$, it is enough to bound the number of k-cuts that respect an arbitrary partition in $\mathcal{F}$ by $n^{o(k)}$. For the digraph D, an integer k and $\mathcal{P} \in \mathcal{F}$, let $N_{\mathcal{P}}$ be the number of k-cuts in D that respect $\mathcal{P}$.

Lemma 10.3.2. *The total number of k-cuts in D is at most $\sum_{\mathcal{P} \in \mathcal{F}} N_{\mathcal{P}}$.*

Proof. To prove the lemma, we need to prove that for any k-cut (L, R) in D, there exists $\mathcal{P} \in \mathcal{F}$ such that (L, R) respects $\mathcal{P}$. This follows from Corollary 10.3.1. □

Henceforth, let us fix $\mathcal{P} = P_1 \uplus \cdots \uplus P_q$, $q \leq 2\lceil\sqrt{k}\rceil$, where $\mathcal{P}$ is an arbitrary partition in $\mathcal{F}$. We are now only interested in bounding the number of k-cuts in D that respect $\mathcal{P}$. It follows from Lemma 10.3.1, that to bound the total number of k-cuts in D, it is sufficient to bound the number of k-cuts in $D[P_i]$, for each $i \in [q]$. We now have the following lemma, that says something much stronger. To bound the number of k-cuts in D that respect $\mathcal{P}$, it is sufficient to bound the number of just the 0-cuts in $D[P_i]$, for all $i \in [q]$.

Lemma 10.3.3. *For any digraph D, let $\mathcal{P} = P_1 \uplus \ldots \uplus P_q$ be some partition of the vertex set of D. For any $i \in [q]$, let N_i be the number of 0-cuts in $D[P_i]$. Then the number of k-cuts in D that respect $\mathcal{P}$ is at most $\prod_{i \in [q]} N_i$.*

Proof. Observe that to prove the lemma it is enough to prove that for any k-cut (L, R) of D that respects $\mathcal{P}$, there exists 0-cuts (L_i, R_i) in $D[P_i]$, for each $i \in [q]$ such that $L = \cup_{i \in q} L_i$ and $R = \cup_{i \in [q]} R_i$. Let (L, R) be some k-cut in D that respects $\mathcal{P}$. For each $i \in [q]$, let $L_i = L \cap P_i$ and $R_i = R \cap P_i$. Observe that, for each $i \in [q]$, (L_i, R_i) is a 0-cut in $D[P_i]$. Suppose not. Then there exists a cut-arc of (L_i, R_i), say (u, v), such that $u, v \in P_i$ and $u \in R_i$, $v \in L_i$. Since $L = \cup_{i \in [q]} L_i$ and $R = \cup_{i \in [q]} R_i$, $u \in R$ and $v \in L$. This contradicts that (L, R) respects $\mathcal{P}$. □

Thus, we have further narrowed down the class of k-cuts that we want to bound. More precisely, we are now interested in bounding the number of 0-cuts in $D[P_i]$, for any part P_i of $\mathcal{P}$. Since $D \in \mathcal{D}_\alpha$, for any $P_i \in \mathcal{P}$, $D[P_i] \in \mathcal{D}_\alpha$. Thus, from

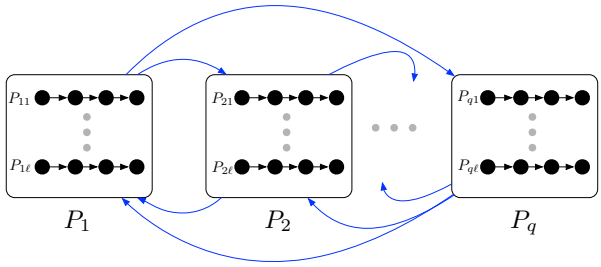


Figure 10.1: The Vertex Partition for the Sub-exponential XP bound. $\mathcal{P} = \{P_1 \uplus \cdots \uplus P_q\}$ is the vertex partition obtained using chromatic coding and $P_i = P_{i1} \uplus \cdots \uplus P_{i\ell}$ is the partition obtained using Gallai-Milgram's Theorem. Each P_{ij} contains a Directed Hamiltonian Path. The cut arcs of all the cuts that respect $\mathcal{P}$ are marked in blue.

Gallai-Milgram Theorem, the vertex set of P_i can be partitioned into at most α parts, say $P_i = P_{i1} \uplus \cdots \uplus P_{i\ell}$, $\ell \leq \alpha$, such that for each $j \in [\ell]$, $D[P_{ij}]$ has a directed Hamiltonian path. We will now prove that for any digraph that has a directed Hamiltonian path, the number of 0-cuts in it are linear in the number of its vertices.

Lemma 10.3.4. *Let D be a digraph on n vertices that has a directed Hamiltonian path. Then the number of 0-cuts in D is $n+1$.*

Proof. Since D has a directed Hamiltonian path, let $\{v_1, \ldots, v_n\}$ be the vertex set of D such that for each $i \in [n-1]$, $(v_i, v_{i+1}) \in E(D)$. Consider any 0-cut (L, R) in D. Let i be the smallest integer such that $v_i \in R$. By the choice of i, for all $j < i$, $v_j \in L$. We now claim that, for all $j > i$, $v_j \in R$. Suppose not. Then there exist a $j > i$, such that $v_j \in L$. Since $j > i$, and v_i appears before v_j in the Hamiltonian path ordering. Thus, there is a directed path from v_i to v_j in D. Since $v_i \in R$ and $v_j \in L$, an arc of this directed path is a cut-arc for (L, R), which contradicts that (L, R) is a 0-cut.

Thus, for any $i \in [n]$, the number of 0-cuts in D where v_i is the first vertex in the ordering $(v_1, \ldots, v_n)$ that belongs to the right part of these cuts is exactly 1. Since any cut in D, either does not contain any vertex in its right part (there is only one such cut) or contains some vertex, the total number of 0-cuts in D is $n+1$. □

We are now ready to prove Lemma 10.1.1. An illustration depicting the partitioning used in the proof of Lemma 10.1.1 is given in Figure 10.1.

Proof of Lemma 10.1.1: Let N be the total number of k-cuts in D. Consider the family $\mathcal{F}$ of Corollary 10.3.1 for the digraph D and integer k. From Corollary 10.3.1, $|\mathcal{F}| \leq 2^{\mathcal{O}(\sqrt{k}\log k)} \cdot \log n$. For each partition $\mathcal{P} \in \mathcal{F}$, let $N_{\mathcal{P}}$ be the number of k-cuts

in D that respect $\mathcal{P}$. From Lemma 10.3.2, $N \leq \sum_{\mathcal{P}\in\mathcal{F}} N_{\mathcal{P}}$. Consider any arbitrary partition $\mathcal{P} \in \mathcal{F}$. Let $\mathcal{P} = P_1 \uplus \ldots \uplus P_q$, and from Corollary 10.3.1 we have $q \leq 2\lceil\sqrt{k}\rceil$. For any $i \in [q]$, let N_{P_i} be the number of 0-cuts in $D[P_i]$. From Lemma 10.3.3, $N_{\mathcal{P}} \leq \prod_{i\in[q]} N_{P_i}$. Since $D \in \mathcal{D}_\alpha$, for any P_i, $D[P_i] \in \mathcal{D}_\alpha$. Thus, from Gallai-Milgram Theorem, the vertex set of P_i can be partitioned into at most α parts, say $P_i = P_{i1} \uplus \ldots \uplus P_{i\ell}$, $\ell \leq \alpha$, such that such that for each $j \in [\ell]$, $D[P_{ij}]$ has a directed Hamiltonian path. From Lemma 10.3.4, the number of 0-cuts in $D[P_{ij}]$ is $n+1$. From Lemma 10.3.1, $N_{P_i} \leq \prod_{j\in[\ell]}(n+1) \leq (n+1)^\ell \leq (n+1)^\alpha$. Combining everything stated above, we get that, $N \leq |\mathcal{F}| \cdot N_{\mathcal{P}} \leq |\mathcal{F}| \cdot (N_{P_i})^{2\lceil\sqrt{k}\rceil} \leq |\mathcal{F}| \cdot (n+1)^{2\alpha\lceil\sqrt{k}\rceil} \leq 2^{\mathcal{O}(\sqrt{k}\log k)} \cdot (n+1)^{2\alpha\lceil\sqrt{k}\rceil} \cdot \log n)$. □

10.4 Improved Bounds on the Number of k-Cuts for Digraphs in $\mathcal{D}_\alpha$ with Bounded Out-Degeneracy

In this section we give the proof of Lemma 10.1.2. Recall from the introduction that a digraph D is said to be d-out-degenerate, if for every subgraph H of D, there exists a vertex $v \in V(H)$, such that $d^+_H(v) \leq d$. Furthermore, a digraph D d-out degenerate if and only if it has a d-out-degeneracy sequence.

Throughout this section, D is a digraph on n vertices and $D \in \mathcal{D}_{\alpha,d}$. Let $(v_1, \ldots, v_n)$ be a d-out-degeneracy sequence of D. For any $i \in [n]$, we say that a k-cut (L, R) in D is of *type-i*, if $v_i \in L$ and for all $j > i$, $v_j \in R$. We say that a k-cut (L, R) in D is of *type-0* if $L = \emptyset$. Note that the collection of the sets of type-i cuts for all $i \in [n]_0$, forms a partition of the set of all the k-cuts. Observe that there is exactly 1 type-0 cut in any digraph.

Observation 10.4.1. *For any $i \in [n]_0$, let N_i be the number of k-cuts in D of type-i. Then the number of k-cuts in D is at most $\sum_{i\in[n]_0} N_i$.*

Henceforth, our goal is to bound the number of k-cuts in D of type-i, for an arbitrary $i \in [n]$. Recall from Lemma 10.3.1 that if $V(D) = V_1 \uplus \cdots \uplus V_c$ is a partition of the vertex set of D, then to bound the number of k-cuts in D, it is enough to bound the number of k-cuts in each $D[V_j]$, $j \in [c]$. This remains our underlying strategy. However, this time we use a different partition of the vertex set of D, where the number of parts of this partition is 4, compared to $o(k)$ in Lemma 10.1.1. This partition of the vertex set, is presented in Lemma 10.4.1.

Lemma 10.4.1. *For a digraph $D \in \mathcal{D}_{\alpha,d}$ and any positive integer k, for any fixed $i \in [n]$, there exists a partition $V(D) = V_{\text{induct}} \uplus V_{\text{forceL}} \uplus V_{\text{forceR}} \uplus V_{\text{small}}$ such that:*

1. *If $\alpha = 1$, then $V_{\text{induct}} = \emptyset$, otherwise $D[V_{\text{induct}}] \in \mathcal{D}_{\alpha',d}$, where $\alpha' < \alpha$.*
2. *For any k-cut (L, R) in D of type-i, $V_{\text{forceL}} \subseteq L$.*
3. *For any k-cut (L, R) in D of type-i, $V_{\text{forceR}} \subseteq R$.*
4. *$|V_{\text{small}}| \leq d + \alpha(2k + 1)$.*

Lemma 10.4.1 states that the vertex set of D can be partitioned into 4 parts with the following properties. The digraph induced on the *first* part is either empty or belongs to $\mathcal{D}_{\alpha',d}$, for $\alpha' < \alpha$. To bound the number of k-cuts in such a digraph we will use an induction on α. For the *second* part of this partition, we prove that for any k-cut (L, R) of type-i, all the vertices of this part belong to L. Similarly, for the *third* part of this partition, we prove that for any k-cut (L, R) of type-i, all the vertices of this part belong to R. Therefore, there is a unique k-cut of type-i in the digraph induced by the second and third part. The *last* part of the partition has the property that the number of vertices in this part is "small". For the digraph induced by this part, we will get the desired bound by using Lemma 10.1.1 on this digraph.

The proof of Lemma 10.4.1 is deferred for later. We will now proceed towards the proof of Lemma 10.1.2 using Lemma 10.4.1 and induction on α. At any inductive step we use the partition of Lemma 10.4.1 and bound the number of k-cuts of type-i in the digraph induced on each part of the partition, thereby bounding the number of k-cuts in D because of Observation 10.4.1.

Proof of Lemma 10.1.2. We prove the lemma using induction on α. For any positive integer α, let us denote the bound of Lemma 10.1.1 on the number of k-cuts in $D \in \mathcal{D}_\alpha$, on at most $d + \alpha(2k + 1)$ vertices, by $\eta(\alpha, d, k)$. That is, $\eta(\alpha, d, k) = 2^{c\sqrt{k}\log k} \cdot (d+1+\alpha(2k+1))^{2\alpha\lceil\sqrt{k}\rceil} \cdot \log(d+\alpha(2k+1))$, where c is the absolute constant hidden in the $\mathcal{O}$ notation of the expression in Proposition 10.3.2. Let $\mathcal{N}_k(n, \alpha, d)$ denote the maximum number of k-cuts in D for any digraph $D \in \mathcal{D}_{\alpha,d}$ on n vertices. We claim that for any positive integers n, d and $\alpha > 1$, $\mathcal{N}_k(n, 1, d) \leq 1 + n \cdot \eta(1, d, k)$ and $\mathcal{N}_k(n, \alpha, d) \leq 1 + \mathcal{N}_k(n, \alpha - 1, d) \cdot \eta(\alpha, d, k) \cdot n$. Solving the recurrence, we will get the desired bound on the number of k-cuts in D.

Let us first prove that for any positive integers n and d, $\mathcal{N}_k(n,1,d) \leq 1 + n \cdot \eta(1,d,k)$. If the independence number of the digraph D is 1, then from Lemma 10.4.1, there exists a partition $V(D) = V_{\text{forceL}} \uplus V_{\text{forceR}} \uplus V_{\text{small}}$ of D such that for any k-cut (L,R) in D of type-i, $V_{\text{forceL}} \subseteq L$ and $V_{\text{forceR}} \subseteq R$. Thus, from Lemma 10.3.1, we conclude that the number of k-cuts of type-i in D is at most the number of k-cuts in $D[V_{\text{small}}]$. Since $D[V_{\text{small}}]$ is an induced subgraph of D, the independence number of $D[V_{\text{small}}]$ is at most α and $D[V_{\text{small}}]$ is a digraph on $d + 2k + 1$ vertices. Thus, we conclude that the number of k-cuts in D of type-i are at most $\eta(1,d,k)$. From Observation 10.4.1, we conclude that the number of k-cuts in D are at most $1 + \eta(1,d,k) \cdot n$.

By induction hypothesis, let us assume that for any positive integers n, d and for all $\alpha' < \alpha$, the number of k-cuts in any digraph $D' \in \mathcal{D}_{\alpha',d}$ on at most n vertices is $\mathcal{N}_k(n,\alpha',d)$. We will now prove that the number of k-cuts in the digraph $D \in \mathcal{D}_{\alpha,d}$ is $\mathcal{N}_k(n,\alpha,d) \leq 1 + \mathcal{N}_k(n,\alpha-1,d) \cdot \eta(\alpha,d,k)$. From Lemma 10.4.1, there exists a partition $V(D) = V_{\text{induct}} \uplus V_{\text{forceL}} \uplus V_{\text{forceR}} \uplus V_{\text{small}}$, such that for any k-cut (L,R) in D of type-i, $V_{\text{forceL}} \subseteq L$ and $V_{\text{forceR}} \subseteq R$. Thus, from Lemma 10.3.1, the number of k-cuts of type-i in D is at most the product of the number of k-cuts in $D[V_{\text{induct}}]$ and the number of k-cuts in $D[V_{\text{small}}]$. Since $D[V_{\text{induct}}] \in \mathcal{D}_{\alpha',d}$, where $\alpha' < \alpha$, from inductive hypothesis we get that the number of k-cuts in $D[V_{\text{induct}}]$ is at most $\mathcal{N}_k(n,\alpha',d) \leq \mathcal{N}_k(n,\alpha-1,d)$. Since $|V_{\text{small}}| \leq d + \alpha(2k+1)$, from Lemma 10.1.1, the number of k-cuts in $D[V_{\text{small}}]$ is at most $\eta(\alpha,d,k)$. Thus, the number of k-cuts of type-i in D is at most $\mathcal{N}_k(n,\alpha-1,d) \cdot \eta(\alpha,d,k)$. From Observation 10.4.1, we conclude that the number of k-cuts in D is at most $1+\mathcal{N}_k(n,\alpha-1,d)\cdot\eta(\alpha,k,d)\cdot n$. □

Proof of Partitioning Lemma. We start by a lemma that gives an upper bound on the size of a digraph in $\mathcal{D}_\alpha$ when every vertex has small out-degree.

Lemma 10.4.2. *For any digraph $D \in \mathcal{D}_\alpha$ and a positive integer k, if for all $v \in V(D)$, $d^+(v) \leq k$, then $|V(D)| \leq \alpha(2k+1)$.*

Proof. Let $|V(D)| = n$. We will first prove that if $D \in \mathcal{D}_\alpha$, then there exists $v \in V(D)$ such that $d^+(v) \geq \frac{(n-\alpha)}{2\alpha}$. Since $d^+(v) \leq k$, for all $v \in V(D)$, this implies that $\frac{(n-\alpha)}{2\alpha} \leq k$, thereby implying that $n \leq \alpha(2k+1)$.

To prove the above-mentioned claim, we invoke Turan's Theorem ([76]), which states that for any graph G and integer r, if G does not contain a clique of size $r+1$, then $|E(G)| \leq (1-\frac{1}{r}) \cdot \frac{|V(G)|^2}{2}$. Let G be the underlying undirected graph of D. Let $\bar{G}$ be the complement graph of G. Since $D \in \mathcal{D}_\alpha$, $\bar{G}$ does not contain a clique of size

$\alpha+1$. Thus, by Turan's Theorem, $|E(\bar{G})| \leq (1-\frac{1}{\alpha}) \cdot \frac{n^2}{2}$. Since $\bar{G}$ is the complement graph of G, $|E(G)| \geq \frac{n(n-1)}{2} - (1-\frac{1}{\alpha}) \cdot \frac{n^2}{2} \geq \frac{(n^2-n\alpha)}{2\alpha}$. Since G is the underlying undirected graph of D, $|E(D)| \geq \frac{(n^2-n\alpha)}{2\alpha}$. Since $|E(D)| = \sum_{v \in V(D)} d^+(v) \geq \frac{(n^2-n\alpha)}{2\alpha}$, there exists $v \in V(D)$, such that $d^+(v) \geq \frac{(n-\alpha)}{2\alpha}$. □

Intuitive Ideas for the proof Lemma 10.4.1. Let us begin by recalling that $(v_1, \ldots, v_n)$ is a d-out-degeneracy sequence of D. Also recall that, the aim of proving Lemma 10.4.1 is to be able to use it to bound the number of k-cuts in D of type-i. Consider any k-cut in D of type-i. By definition, $v_i \in L$ and for all $j > i$, $v_j \in R$. Thus, $v_i \in V_{\text{forceL}}$ and $\{v_j \mid j > i\} \subseteq V_{\text{forceR}}$. Thus, to prove Lemma 10.4.1, we essentially need to partition the vertices that appear before v_i in $(v_1, \ldots, v_n)$. Consider the non-neighbors of v_i. They induce a digraph whose independence number is strictly less than the independence number of D. Thus, they go to V_{induct}. Thus, we are now left with the goal of partitioning the set of neighbors of v_i that appear before v_i in $(v_1, \ldots, v_n)$. Since $(v_1, \ldots, v_n)$ is a d-out-degeneracy sequence of D, the number of out-neighbors of v_i that appear before v_i in $(v_1, \ldots, v_n)$ is at most d. This set of neighbors goes to the set V_{small}. Finally, we are left with the set, say X, of vertices that appear before v_i in $(v_1, \ldots, v_n)$ and are in-neighbors of v_i. Here, we observe that, if any vertex $v \in X$ has out-degree at least $k+1$ in $D[X]$, then there are at least $k+1$ arc-disjoint paths from v to v_i in $D[X \cup \{v_i\}]$, and hence in D. Thus, such a vertex v should always belong to same part as v_i in any k-cut. Thus, such vertices goes to V_{forceL}. Finally, the remaining vertex set, say X', has the property that each vertex in X has out-degree at most k. By Lemma 10.4.2, in such a case the size of X' is at most $\alpha(2k+1)$, and hence X' goes to V_{small}. We are now ready to prove Lemma 10.4.1 formally.

Proof of Lemma 10.4.1. Let $(v_1, \ldots, v_n)$ be a d-out-degeneracy sequence of D. Consider the partition of $V(D)$ into three parts: $\{v_i\}$, the predecessors of v_i in this ordering, V_P and the successors of v_i in this ordering V_S. Formally, consider $V(D) = \{v_i\} \uplus V_P \uplus V_S$, where $V_P = \{v_j : j < i\}$ and $V_S = \{v_j : j > i\}$. Further consider the partition of V_P into the set of vertices of V_P that are neighbors of v_i, say V_P^N, and the set of vertices of V_P that are non-neighbors of v_i, say V_P^{NN}. That is, $V(P) = V_P^N \uplus V_P^{NN}$. Next consider the partition of V_P^N into two parts: V_P^{ON} and V_P^{IN} such that V_P^{ON} is the set of vertices in V_P^N that are out-neighbors of v_i and V_P^{IN} is the set of vertices in V_P^N that are in-neighbors of v_i. Finally, consider the digraph induced on V_P^{IN}. We partition the set V_P^{IN} based on the out-degree of the vertices in $D' = D[V_P^{IN} \cup \{v_i\}]$. We partition the set V_P^{IN} into two parts: $V_{P,L}^{IN}$ and $V_{P,S}^{IN}$, in the following way. If $d^+_{D'}(v) \geq k+1$, $v \in V_{P,L}^{IN}$, otherwise $v \in V_{P,S}^{IN}$. Observe that, for

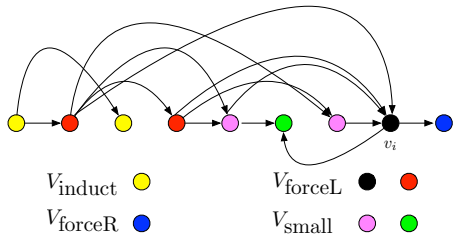


Figure 10.2: The vertex partition for the Sub-exponential FPT bound. Here the vertices are arranged in the linear order respecting the d-out-degeneracy sequence of D. Here $k = 2$ and the partition of the vertices into the respective sets is demonstrated using appropriate colors.

each $v \in V_{P,S}^{IN}$, $d^+_{D''(v)} \leq k$, where $D'' = D[V_{P,S}^{IN} \cup \{v_i\}]$. We have the following from the above discussion.

$$V(D) = \{v_i\} \uplus V_P \uplus V_S = \{v_i\} \uplus V_P^N \uplus V_P^{NN} \uplus V_S = \{v_i\} \uplus V_P^{ON} \uplus V_P^{IN} \uplus V_P^{NN} \uplus V_S$$
$$= \{v_i\} \uplus V_P^{ON} \uplus V_{P,L}^{IN} \uplus V_{P,S}^{IN} \uplus V_P^{NN} \uplus V_S.$$

We now claim that the desired partition $V(D) = V_{\text{induct}} \uplus V_{\text{forceL}} \uplus V_{\text{forceR}} \uplus V_{\text{small}}$ is such that, (1) $V_{\text{induct}} = V_P^{NN}$, (2) $V_{\text{forceL}} = \{v_i\} \cup V_{P,L}^{IN}$, (3) $V_{\text{forceR}} = V_S$, and (4) $V_{\text{small}} = V_P^{ON} \cup V_{P,S}^{IN}$. An illustration depicting this partitioning can be found in Figure 10.2. Let us now prove that the sets $V_{\text{induct}}, V_{\text{forceL}}, V_{\text{forceR}}$ and V_{small} satisfy the desired properties.

1. V_{induct}**:** Observe that when $\alpha = 1$, that is, when D is a tournament, $V_P^{NN} = \emptyset$. Therefore, in this case, $V_{\text{induct}} = \emptyset$. Otherwise, since $D[V_{\text{induct}}]$ is a subgraph of D and $D \in \mathcal{D}_{\alpha,d}$, $D[V_{\text{induct}}] \in \mathcal{D}_{\alpha,d}$. Since V_{induct} only contains vertices that are non-neighbors of v_i, if $D[V_{\text{induct}}]$ has an independent set, say X, of size α then $X \cup \{v_i\}$ is an independent set in D of size $\alpha + 1$, which contradicts the fact that the size of any independent set in D is bounded by α. Thus, $D[V_{\text{induct}}] \in \mathcal{D}_{\alpha',d}$, where $\alpha' < \alpha$.

2. V_{forceL}**:** By the definition of a type-i cut, for any k-cut (L, R) of type-i in D, $v_i \in L$. We will now show that for any $v_j \in V_{P,L}^{IN}$, there exists $k+1$ arc-disjoint paths from v_j to v_i. Thus, if (L, R) is a k-cut in D and $v_i \in L$, then for all $v_j \in V_{P,L}^{IN}$, $v_j \in L$. Consider any $v_j \in V_{P,L}^{IN}$. Recall that $d^+_{D'}(v_j) \geq k + 1$ where $D' = D[V_P^{IN} \cup \{v_i\}]$ and V_P^{IN} is the set of in-neighbors of v_i in V_P. Consider the set of out-neighbours of v_j in D'. Since the number of such out-neighbors

is at least $k+1$ and each of these out-neighbors is an in-neighbor of v_i, we conclude that there are at least $k+1$ arc-disjoint paths from v_j to v_i.

3. V_{forceR}**:** By the definition of type-i cut, $V_S \subseteq R$, for any type-i cut (L, R).

4. V_{small}**:** Since $(v_1, \ldots, v_n)$ is a d-out-degeneracy sequence of D, $|V_P^{NO}| \leq d$. We need to show that $|V_{P,S}^{IN}| \leq \alpha(2k+1)$. Recall that, as observed before, for each $v \in V_{P,S}^{IN}$, $d^+_{D''}(v) \leq k$, where $D'' = D[V_{P,S}^{IN} \cup \{v_i\}]$. Since D'' is an induced subgraph of D, $D'' \in \mathcal{D}_{\alpha,d}$. Also for each $v \in V(D'')$, $d^+_{D''}(v) \leq k$. Thus, from Lemma 10.4.2, $|V(D'')| \leq \alpha(2k+1)$. This proves that $|V_{P,S}^{IN}| \leq \alpha(2k+1)$.

This concludes the proof. □

10.5 Sub-Exponential FPT Algorithms for DFAS, Directed Cutwidth and OLA for Digraphs in $\mathcal{D}_\alpha$

In this section, we will give sub-exponential FPT algorithms for DFAS, Directed Cutwidth and Optimal Linear Arrangement when the input graph belongs to $\mathcal{D}_\alpha$, for some positive integer α. All these algorithms are based on a three step procedure. The *first* is observing that the digraphs that are YESinstances of these problems have *sub-exponential FPT* many k-cuts. The proofs for DFAS and Directed Cutwidth are based on showing that the digraph in the Yes instances of the problems are k-out-degenerate, and hence, the bounds follow from Lemma 10.1.2. For OLA, we show that if there is an ordering of the vertex set of a digraph of cost at most k then the cutwidth of this digraph is $\mathcal{O}(k^{2/3})$. Hence, from the results for Directed Cutwidth, the number of k-cuts in the Yes instances of OLA is also bounded. The *second* step is a procedure to enumerate all k-cuts of the input digraph. And the *third* is to do some dynamic programming procedure over these enumerated cuts to solve the respective problems. The last part of the algorithm (doing dynamic programming over k-cuts) is standard and is identical to the algorithm given by Fomin and Pilipczuk [102]. Proofs are given for completeness.

Before proceeding further, we make a small remark that the proofs of Lemma 10.1.1 and 10.1.2 can be made constructive by using the constructive versions of the Gallai-Milgram's Theorem, Chromatic Coding lemma and a polynomial time procedure to output a d-out-degeneracy sequence of a digraph. Thus, one can actually enumerate

all the k-cuts in the input digraphs of these Lemmas using our algorithm. However, for the sake of completeness, we state in Lemma 10.5.1, a different procedure that using a standard branching, enumerates all the k-cuts in any digraph with polynomial delay.

Lemma 10.5.1 (Lemma 7, [102])**.** *k-cuts of a digraph D can be enumerated with polynomial-time delay.*

10.5.1 Sub-Exponential Algorithm for DFAS

Let (D, k) be an instance of DFAS. Any solution S to this instance is called a *dfas* of D. Observe that, a digraph D has a *dfas* of size at most k if and only if there exists an ordering, say $(v_1, \ldots, v_n)$, of $V(D)$ such that $|\sum_{i\in[n]} N^+(v_i) \cap \{v_j \mid j < i\}| \leq k$, that is, the number of backward arcs in this ordering is at most k. Below we bound the number of k-cuts in the YES instances of DFAS.

Lemma 10.5.2. *If (D, k) is a* YES *instance of* DFAS *and $D \in \mathcal{D}_\alpha$, then the number of k-cuts in D is at most $2^{c(\alpha+1)\sqrt{k}\log k} \cdot 2^{2\alpha(\alpha+1)\lceil\sqrt{k}\rceil \log((k(2\alpha+1)+\alpha+1))} \cdot \log(k + \alpha(2k + 1)) \cdot n^{\alpha+1}$, where c is a fixed absolute constant.*

Proof. Since (D, k) is a YES instance of DFAS, there exists an ordering, say $(v_1, \ldots, v_n)$, of $V(D)$, such that $|\sum_{i\in[n]} N^+(v_i) \cap \{v_j \mid j < i\}| \leq k$. In particular, for any $i \in [n]$, $|N^+(v_i) \cap \{v_j \mid j < i\}| \leq k$. Thus, $(v_1, \ldots, v_n)$ is a k-out-degeneracy sequence of $V(D)$. Therefore, the bound follows from Lemma 10.1.2. □

Now we give the proof of Theorem 10.0.1.

Proof of Theorem 10.0.1. Using the algorithm of Lemma 10.5.1, we enumerate all k-cuts in D. If during the enumeration we exceed the bound given in Lemma 10.5.2, then we correctly conclude that (D, k) is a NO instance of DFAS. Otherwise, from Lemma 10.5.1, in time $2^{\mathcal{O}(\alpha^2\sqrt{k}\log(\alpha k))} \cdot n^{\mathcal{O}(\alpha)}$, we would have enumerated the set of all k-cuts in D. Let us denote this set by $\mathcal{C}$. We will solve the DFAS problem by doing a dynamic programming over the set $\mathcal{C}$ of k-cuts. Let T be the dynamic programming table indexed by a k-cut $(L, R) \in \mathcal{C}$ and an integer $i \in [k]$. For any $(L, R) \in \mathcal{C}$ and $i \in [k]$, we want $T((L, R), i)$ to store the following information.

$$T((L,R),i) = \begin{cases} 1 & \text{if there exists an ordering } (v_1,\ldots,v_\ell) \text{ of } L \\ & \text{witnessing that } D[L] \text{ has a } \textit{dfas} \text{ of size } i, \text{ and} \\ & (L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$$

Note that $T((V(D),\emptyset),k) = 1$ if and only if D has a *dfas* of size at most k. We now describe how we compute $T((L,R),i)$, for any $(L,R) \in \mathcal{C}$ and $i \in [k]$. For all $i \in [k]$, $T((\emptyset,V(D)),i) = 1$. For any $(L,R) \in \mathcal{C}$, such that $L \neq \emptyset$, and any $i \in [k]$, $T((L,R),i) = 1$ if and only if there exists $v \in L$ such that $(L \setminus \{v\}, R \cup \{v\}) \in \mathcal{C}$ and, if $|N_L^+(v)| = j$, then $T((L \setminus \{v\}, R \cup \{v\}), i-j) = 1$.

We now prove that for any $(L,R) \in \mathcal{C}$ and $i \in [k]$, $T((L,R),i) = 1$ if and only if there exists an ordering $(v_1,\ldots,v_\ell)$ of L witnessing that $D[L]$ has a *dfas* of size i, and $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$. We prove this by induction on $|L|$. When $|L| = 0$, this is true because of the base case. By inductive hypothesis, assume that it holds for any $(L',R') \in \mathcal{C}$ such that $|L'| = \ell - 1$, and for any $i \in [k]$. We will first prove that if $T((L,R),i) = 1$, then there exists an ordering $(v_1,\ldots,v_\ell)$ of L witnessing that $D[L]$ has a *dfas* of size i, and $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$.

Since $T((L,R),i) = 1$, there exists a vertex, say $v_\ell \in L$, such that $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$ and if $|N_L^+(v_\ell)| = j$ then $T((L \setminus \{v_\ell\}, R \cup \{v_\ell\}), i-j) = 1$. Since $T((L \setminus \{v_\ell\}, R \cup \{v_\ell\}), i-j) = 1$, from induction hypothesis, $D[L \setminus \{v_\ell\}]$ has a *dfas* of size at most $i-j$. Let $(v_1,\ldots,v_{\ell-1})$ be the ordering of $L \setminus \{v_\ell\}$ witnessing this, that is, $\sum_{p \in [\ell-1]} |N^+(v_p) \cap \{v_q \mid q < p\}| \leq i-j$. Since $|N_L^+(v_\ell)| = j$, $\sum_{p \in [\ell]} |N^+(v_p) \cap \{v_q \mid q < p\}| \leq i$. Thus, the ordering $(v_1,\ldots,v_{\ell-1},v_\ell)$ is a witness to the fact that $D[L]$ has a *dfas* of size at most i.

We will now prove that if $D[L]$ has a *dfas* of size at most i and $(v_1,\ldots,v_\ell)$ is an ordering witnessing this such that $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$, then $T((L,R),i) = 1$. Clearly, if $|N^+(v_\ell)| = j$, then the ordering $(v_1,\ldots,v_{\ell-1})$ witnesses that $D[L \setminus \{v_\ell\}]$ has a *dfas* of size at most $i-j$. Thus, $T((L \setminus \{v_\ell\}, R \cup \{v_\ell\}), i-j) = 1$.

□

10.5.2 Sub-Exponential Algorithm for Directed Cutwidth

Let D be a digraph. For an ordering $(v_1, \ldots, v_n)$ of $V(D)$, the *width* of this ordering is $\max_{i \in [n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})|$. The *cutwidth* of D, denoted by $\mathbf{ctw}(D)$, is the smallest possible width of an ordering of $V(D)$.

Directed Cutwidth	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Is $\mathbf{ctw}(D) \leq k$?	

Next we bound the number of k-cuts in the Yes instances of DFAS.

Lemma 10.5.3. *If (D, k) is a* Yes *instance of* Directed Cutwidth *and $D \in \mathcal{D}_\alpha$, then the number of k-cuts in D is at most $2^{c(\alpha+1)\sqrt{k}\log k} \cdot 2^{2\alpha(\alpha+1)\lceil\sqrt{k}\rceil \log((k(2\alpha+1)+\alpha+1))} \cdot \log(k+\alpha(2k+1)) \cdot n^{\alpha+1}$, where c is a fixed absolute constant.*

Proof. If (D, k) is a Yes instance of DFAS, then there is an ordering, say $(v_1, \ldots, v_n)$, of $V(D)$ of width at most k. Recall that, the width of an ordering $(v_1, \ldots, v_n)$ is $\max_{i \in [n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})|$. Observe that if $\max_{i \in [n-1]} |E(\{v_1, \ldots, v_i\}, \{v_{i+1}, \ldots, v_n\})| \leq k$, then for each $i \in [n]$, $|N^+(v_i) \cap \{v_j : j < i\}| \leq k$. Thus, D is k-out-degenerate. Thus, the bound follows from Lemma 10.1.2. □

Now we give the proof of Theorem 10.0.2.

Proof of Theorem 10.0.2. Using the algorithm of Lemma 10.5.1, we enumerate all k-cuts in D. If during the enumeration we exceed the bound given in Lemma 10.5.3, then we correctly conclude that (D, k) is a No instance of Directed Cutwidth. Otherwise, from Lemma 10.5.1, in time $2^{\mathcal{O}(\alpha^2\sqrt{k}\log(\alpha k))} \cdot n^{\mathcal{O}(\alpha)}$, we would have enumerated the set of all k-cuts in D. Let us denote this set by $\mathcal{C}$. We will solve the Directed Cutwidth problem by doing a dynamic programming over the set $\mathcal{C}$ of k-cuts. Let T be the dynamic programming table indexed by a k-cut $(L, R) \in \mathcal{C}$. For any $(L, R) \in \mathcal{C}$, we want $T((L, R))$ to store the following information.

$$T((L,R)) = \begin{cases} 1 & \text{if there exists an ordering of } L, \text{ say } (v_1, \ldots, v_\ell), \\ & \text{such that for all } j \in [\ell-1], |E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq k \\ 0 & \text{otherwise} \end{cases}$$

Note that $T((V(D), \emptyset)) = 1$ if and only if $\mathbf{ctw}(D) \leq k$. We now describe how we compute $T((L, R))$ for any $(L, R) \in \mathcal{C}$. Set $T((\emptyset, V(D))) = 1$. For any $(L, R) \in \mathcal{C}$ such that $L \neq \emptyset$, $T((L, R)) = 1$ if and only if there exists $v \in L$ such that $(L \setminus \{v\}, R \cup \{v\}) \in \mathcal{C}$ and $T((L \setminus \{v\}, R \cup \{v\})) = 1$.

We now prove that for any $(L, R) \in \mathcal{C}$, $T((L, R)) = 1$ if and only if there exists an ordering of L, say $(v_1, \ldots, v_\ell)$, such that for all $j \in [\ell - 1]$, $|E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq k$. We prove this by induction on $|L|$. When $|L| = 0$, this is true because of the base case. By inductive hypothesis, assume that for any $(L', R') \in \mathcal{C}$, such that $|L'| = \ell - 1$, $T((L', R')) = 1$ if and only if there exists an ordering of L', say $(v_1, \ldots, v_{\ell-1})$, such that for all $j \in [\ell - 2]$, $|E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq k$. Let $(L, R) \in \mathcal{C}$ be such that $|L| = \ell$. We will first prove that if $T((L, R)) = 1$, then there exists an ordering of L, say $(v_1, \ldots, v_\ell)$, such that for all $j \in [\ell - 1]$, $|E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq k$. Since $T((L, R)) = 1$, there exists a vertex in L, say v_ℓ, such that $(L \setminus \{v_\ell\}, R \cup \{v_\ell\})$ and $T((L \setminus \{v_\ell\}, R \cup \{v_\ell\})) = 1$. Since $T((L \setminus \{v_\ell\}, R \cup \{v_\ell\})) = 1$, from inductive hypothesis, there exists an ordering of $L \setminus \{v_\ell\}$, say $(v_1, \ldots, v_{\ell-1})$, such that for all $j \in [\ell - 2]$, $|E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq k$. Also, since $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$, $|E(\{v_\ell, \ldots, v_n\}, \{v_1, \ldots, v_{\ell-1}\})| \leq k$. Thus, for the ordering $(v_1, \ldots, v_\ell)$ of L, for all $j \in [\ell - 1]$, $|E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq k$.

We will now prove that if there exists an ordering of L, say $(v_1, \ldots, v_\ell)$, such that for all $j \in [\ell - 1]$, $|E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq k$, then $T((L, R)) = 1$. Since $|E(\{v_\ell, \ldots, v_n\}, \{v_1, \ldots, v_{\ell-1}\})| \leq k$, $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$. Also, since for all $j \in [\ell - 2]$, $|E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq k$, therefore, $T((L \setminus \{v_\ell\}, R \cup \{v_\ell\})) = 1$. Thus, $T((L, R)) = 1$. This concludes the proof.

□

10.5.3 Sub-Exponential Algorithm for OLA

Let D be a digraph. For an ordering $\sigma = (v_1, \ldots, v_n)$ of $V(D)$, the *cost* of σ is

$$\sum_{(v_i, v_j) \in E(D)} (i - j) \cdot [i > j],$$

that is, every arc directed backward in the ordering contributes a cost that is equal to the length of this arc, which is the distance between the end-points of this arc in the ordering. Recall that $[i > j]$, evaluates to 1 if $i > j$, to 0 otherwise.

Optimal Linear Arrangement (OLA)	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Is there an ordering of $V(D)$ of cost at most k?	

The following proposition gives an alternate definition of the cost of an ordering.

Proposition 10.5.1 ([102]). *For a digraph D and an ordering $(v_1, \ldots, v_n)$ of $V(D)$, the cost of this ordering is equal to $\sum_{i\in[n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})|$.*

Lemma 10.5.4 shows a relation between the cost of an ordering and its width. Note that this lemma was already proved in [102], but the authors state the result for the case when the input digraph is a semi-complete digraph. We observe that the same proof works for any digraph. For the sake of completeness, we give the same proof here.

Lemma 10.5.4. *For any digraph D, if there is an ordering say $(v_1, \ldots, v_n)$ of $V(D)$, of cost at most k, then $\mathbf{ctw}(D) \leq (2k)^{\frac{2}{3}}$.*

Proof. Since $(v_1, \ldots, v_n)$ is an ordering of cost at most k, from Proposition 10.5.1, $\sum_{i\in[n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})| \leq k$. Fix an arbitrary $i \in [n-1]$. We will show that $|E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})| \leq (2k)^{\frac{2}{3}}$. Let $|E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})| = \ell$. For any arc $(v_p, v_q) \in E(D)$, such that $p < q$, the length of the arc (v_p, v_q) is equal to $q - p$. Observe that, for any r, the number of arcs of length exactly r with tail in $\{v_{i+1}, \ldots, v_n\}$ and head in $\{v_1, \ldots, v_i\}$ is at most r. Thus, for any r, the total number of arcs of length at most r, with tail in $\{v_{i+1}, \ldots, v_n\}$ and head in $\{v_1, \ldots, v_i\}$, is at most $\frac{r(r+1)}{2}$. In particular, the number of arcs of length at most $\sqrt{\ell} - 1$, with tail in $\{v_{i+1}, \ldots, v_n\}$ and head in $\{v_1, \ldots, v_i\}$ is at most $\frac{\sqrt{\ell}(\sqrt{\ell}-1)}{2} \leq \frac{\ell}{2}$. Since $|E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})| = \ell$, the number of arcs of length at least $\sqrt{\ell}$ with tail in $\{v_{i+1}, \ldots, v_n\}$ and head in $\{v_1, \ldots, v_i\}$ is at least $\frac{\ell}{2}$. Since $\sum_{i\in[n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})| \leq k$, we have that $k \geq \sqrt{\ell} \cdot \frac{\ell}{2}$. Thus, $\ell \leq (2k)^{\frac{2}{3}}$. □

Next we bound the number of k-cuts in the Yes instances of OLA.

Lemma 10.5.5. *If (D, k) is a* Yes *instance of* OLA *and $D \in \mathcal{D}_\alpha$, then the number of k-cuts in D is at most $2^{c(\alpha+1)k^{\frac{1}{3}}\log k} \cdot 2^{2\alpha(\alpha+1)\lceil k^{\frac{1}{3}}\rceil \log((k(2\alpha+1)+\alpha+1))} \cdot \log(k + \alpha(2k+1)) \cdot n^{\alpha+1}$, where c is a fixed absolute constant.*

Proof. Since D is a YES instance of OLA, from Lemma 10.5.4, $\mathbf{ctw}(D) \leq (2k)^{\frac{2}{3}}$. Thus, $(D, (2k)^{\frac{2}{3}})$ is a YESinstance of DIRECTED CUTWIDTH. Hence, from Lemma 10.5.3, the number of k-cuts in D are bounded by the desired function. □

Proof of Theorem 10.0.3. Using the algorithm of Lemma 10.5.1, we enumerate all k-cuts in D. If during the enumeration we exceed the bound given in Lemma 10.5.5, then we correctly conclude that (D, k) is a NO instance of OLA. Otherwise, from Lemma 10.5.1, in time $2^{\mathcal{O}(\alpha^2 k^{\frac{1}{3}} \log(\alpha k))} \cdot n^{\mathcal{O}(\alpha)}$, we would have enumerated the set of all k-cuts in D. Let us denote this set by $\mathcal{C}$. We will solve OLA by doing a dynamic programming over the set $\mathcal{C}$ of k-cuts. Let T be the dynamic programming table indexed by a k-cut $(L, R) \in \mathcal{C}$ and an integer $i \in [k]$. For any $(L, R) \in \mathcal{C}$ and $i \in [k]$, we want $T((L, R), i)$ to store the following information.

$$T((L,R),i) = \begin{cases} 1 & \text{if there exists an ordering of } L, \text{ say } (v_1, \ldots, v_\ell), \\ & \text{such that } \sum_{j \in [\ell]} |E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq i \\ 0 & \text{otherwise} \end{cases}$$

Note that $T((V(D), \emptyset), k) = 1$ if and only if D has an ordering of cost at most k. We now describe how we compute $T((L, R), i)$ for any $(L, R) \in \mathcal{C}$ and $i \in [k]$. For all $i \in [k]$, $T((\emptyset, V(D)), i) = 1$. For any $(L, R) \in \mathcal{C}$ such that $L \neq \emptyset$, and any $i \in [k]$, $T((L, R)) = 1$ if and only if there exists $v \in L$ such that $(L \setminus \{v_\ell\}, R \cup \{v_\ell\})$ and $T((L \setminus \{v\}, R \cup \{v\}), i - j) = 1$, where $j = |E(R, L)|$.

We now prove that for any $(L, R) \in \mathcal{C}$ and integer $i \in [k]$, $T((L, R), i) = 1$ if and only if there exists an ordering of L, say $(v_1, \ldots, v_\ell)$, such that $\sum_{j \in [\ell]} |E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq i$. We prove this by induction on $|L|$. When $|L| = 0$, this is true because of the base case. By inductive hypothesis, assume that for any $(L', R') \in \mathcal{C}$ such that $|L'| = \ell - 1$, and for any $p \in [k]$, $T((L', R'), p) = 1$ if and only if there exists an ordering of L, say $(v_1, \ldots, v_\ell)$, such that $\sum_{j \in [\ell]} |E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq i$. Let $(L, R) \in \mathcal{C}$ be such that $|L| = \ell$ and $i \in [k]$. We will first prove that if $T((L, R), i) = 1$, then there exists an ordering of L, say $(v_1, \ldots, v_\ell)$, such that $\sum_{j \in [\ell]} |E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq i$. Let $j = |E(R, L)|$. Since $T((L, R), i) = 1$, there exists a vertex in L, say v_ℓ, such that $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$ and $T((L \setminus \{v_\ell\}, R \cup \{v_\ell\}), i - j) = 1$. From inductive hypothesis, there exists an ordering of $L \setminus \{v_\ell\}$, say $(v_1, \ldots, v_{\ell-1})$, such that $\sum_{p \in [\ell-1]} |E(\{v_{p+1}, \ldots, v_n\}, \{v_1, \ldots, v_p\})| \leq i - j$. Since $j = |E(R, L)|$, for the ordering $(v_1, \ldots, v_\ell)$ of L, $\sum_{p \in [\ell]}$

$|E(\{v_{p+1}, \ldots, v_n\}, \{v_1, \ldots, v_p\})| \leq i$.

We will now prove that if there exists an ordering of L, say $(v_1, \ldots, v_\ell)$, such that $\sum_{j \in [\ell]} |E(\{v_{j+1}, \ldots, v_n\}, \{v_1, \ldots, v_j\})| \leq i$, then $T((L, R), i) = 1$. Observe from the definition of this ordering $(v_1, \ldots, v_\ell)$ that $(L \setminus \{v_\ell\}, R \cup \{v_\ell\})$ is an i-cut in D. Since $i \leq k$, $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$. Clearly, if $|E(R, L)| = j$, then $\sum_{p \in [\ell-1]} |E(\{v_{p+1}, \ldots, v_n\}, \{v_1, \ldots, v_p\})| \leq i-j$. Thus, $T((L \setminus \{v_\ell\}, R \cup \{v_\ell\}), i-j) = 1$ implying that $T((L, R), i) = 1$. This concludes the proof. □

Chapter 11

Improved Sub-Exponential FPT Algorithms for DFAS and DEOCT on Bounded Independence Number Digraphs

In this chapter, we improve the sub-exponential FPT algorithm for DFAS on bounded independence number digraphs that was designed in Chapter 10 by omitting the dependence of α (recall α is the size of the largest independent set in the input digraph) from the exponent of n. We also design a sub-exponential FPT algorithm for DEOCT on digraphs of bounded independence number. Formally, we prove the following theorems.

Theorem 11.0.1. DFAS *on* $\mathcal{D}_\alpha$ *can be solved in* $2^{f(\alpha)\sqrt{k}\log k} \cdot n^{\mathcal{O}(1)}$*, where* $f(\alpha)$ *is some function of* α *and* n *is the number of vertices in* D*.*

Theorem 11.0.2. DEOCT *on* $\mathcal{D}_\alpha$ *admits an algorithm with running time* $2^{\mathcal{O}(f(\alpha)\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$*, where* $f(\alpha)$ *is a function of* α *and* n *is the number of vertices in* D*.*

Recall from Chapter 10 that a *k-cut* of a digraph D is a partition of the vertex set of D into two parts, $V(D) = L \uplus R$, such that $|E(R, L)| \leq k$. Also recall the bound of Lemma 10.1.1 on the number of k-cuts in any digraph $D \in \mathcal{D}_\alpha$. Further, the k-cuts in any digraph can be enumerated by polynomial delay (Lemma 10.5.1).

11.1 Improved Sub-Exponential FPT Algorithm for DFAS on $\mathcal{D}_\alpha$.

A sub-exponential FPT algorithm for DFAS was presented in Chapter 10 with running time $2^{\alpha^2\sqrt{k}\log(\alpha k)}n^{\mathcal{O}(\alpha)}$. This algorithm is obtained by a dynamic programming on the number of k-cuts in an input instance (D,k). The above running time directly follows from the number of k-cuts in a digraph $D \in D_\alpha$ of bounded out-degeneracy (Lemma 10.1.2). We can obtain a faster algorithm by first applying Theorem 3.1.1 to the input instance (D,k) to obtain a kernel (D',k'), and then applying Lemma 10.1.1 to (D',k'). This procedure gives a running time of $2^{\mathcal{O}(f(\alpha)\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$ for some function f, of α), thereby proving Theorem 11.0.1.

11.2 Sub-Exponential FPT Algorithm for DEOCT on $\mathcal{D}_\alpha$

In this section, we prove Theorem 11.0.2.

Theorem 11.0.2. DEOCT *on* $\mathcal{D}_\alpha$ *admits an algorithm with running time* $2^{\mathcal{O}(f(\alpha)\sqrt{k}\log k)}$. $n^{\mathcal{O}(1)}$, *where* $f(\alpha)$ *is a function of* α *and* n *is the number of vertices in* D.

Our approach is similar to the design of algorithms in Chapter 10, but requires some additional work to handle the even cycles that remain after removing a solution.

For a digraph D, let us define a γ-*vertex sequence* of D as a sequence of vertex sets of D say $(C_1,\ldots,C_t)$, such that,

1. for all $i,j \in [t]$, $i \neq j$, $C_i \cap C_j = \emptyset$ and $C_1 \uplus \ldots \uplus C_t = V(D)$, and
2. for all $i \in [t]$, $|C_i| \leq \gamma$,

For any subset $C_i \subseteq V(D)$, *deoct*(C_i) denotes the size of the minimum *deoct* of $D[C_i]$. The *cost* of a γ-vertex sequence $(C_1,\ldots,C_t)$ of D is defined as $\sum_{i\in[t]} deoct(C_i) + |\{(u,v) : (u,v) \in E(D), u \in C_j, v \in C_i, j > i\}|$. For the rest of the section, fix $\gamma = \alpha + \sqrt{\alpha^2 + 8\alpha k}$.

Lemma 11.2.1. *Let* $D \in \mathcal{D}_\alpha$. *For any positive integer* k, (D,k) *is* YES *instance of* DEOCT *if and only if there exists a* γ*-vertex sequence of* D *of cost at most* k.

Proof. For the forward direction, let (D, k) be a YES instance of DEOCT. Let S be a *deoct* of D of size at most k. Consider the digraph $D - S$. Note that the vertex set of $D - S$ is the same as the vertex set of D. Let $(C_1, \ldots, C_t)$ be the topological ordering of the strongly connected components of $D - S$, that is, each C_i is a strongly connected component of $D - S$ and if there exists $(u, v) \in E(D - S)$, such that $u \in C_j$ and $v \in C_i$, then $j > i$. We will now show that $(C_1, \ldots, C_t)$ is a γ-vertex sequence of D of cost at most k. First observe that $C_1 \uplus \ldots C_t = V(D)$. We will now show that for each $i \in [t]$, $|C_i| \leq \gamma$ and cost of $(C_1, \ldots, C_t)$ is at most k.

Claim 11.2.1. *For any $i \in [t]$, $|C_i| \leq \gamma$.*

Proof. Since C_i is a strongly connected component of $D - S$ and S is a *deoct* of D, from Proposition 4.4.1 C_i is a bipartite graph in $D - S$. Let (A_i, B_i) be a bipartition of C_i. We will now show that $|A_i|, |B_i| \leq \frac{\gamma}{2}$. This will prove the claim. Let us argue that $|A_i| \leq \frac{\gamma}{2}$, the other case is symmetric. Since $D[A_i]$ is a subgraph of D, $D[A_i] \in \mathcal{D}_\alpha$. Thus, from Lemma 3.3.1, $E(D[A_i]) \geq \frac{|A_i|^2}{2\alpha} - \frac{|A_i|}{2}$. Since, A_i is an independent set in $D - S$, $|S| \geq E(D[A_i]) \geq \frac{|A_i|^2}{2\alpha} - \frac{|A_i|}{2}$. Then if $|A_i| > \frac{\gamma}{2}$, the we have that $|S| > k$, which is a contradiction. The same argument holds for B_i too. Thus, we conclude that $|C_i| \leq \gamma$. □

Claim 11.2.2. *The cost of $(C_1, \ldots, C_t)$ is at most k.*

Proof. To show this, we will prove that the cost of $(C_1, \ldots, C_t)$ is at most $|S|$. Recall that cost of $(C_1, \ldots, C_t)$ is $\sum_{i \in [t]} \mathit{deoct}(C_i) + |\{(u, v) : (u, v) \in E(D), u \in C_j, v \in C_i, j > i\}|$. Let us denote $E_{back} = \{(u, v) : (u, v) \in E(D), u \in C_j, v \in C_i, j > i\}$. Since $(C_1, \ldots, C_t)$ is a topological ordering of the strongly connected components of $D - S$, $E_{back} \subseteq S$. Also, for any $i \in [t]$, $|\mathit{deoct}(C_i)| \leq |S \cap E(D[C_i])|$. Thus, cost of $(C_1, \ldots, C_t)$ is at most $|S|$. □

Claims 11.2.1 and 11.2.2 prove the forward direction of the lemma. We now prove the backward direction. Let $(C_1, \ldots, C_t)$ be a γ-vertex sequence of D of cost at most k. Let $E_{back} = \{(u, v) : (u, v) \in E(D), u \in C_j, v \in C_i, j > i\}$. We will now show that $S = \cup_{i \in [t]} \mathit{deoct}(C_i) \cup E_{back}$ is a *deoct* of D. Observe that $|S|$ is equal to the cost of $(C_1, \ldots, C_t)$. Suppose S is not a *deoct* of D. Then there exists an odd cycle in $D - S$. Since, for all $i \in [t]$, $\mathit{deoct}(C_i) \subseteq S$, such a cycle cannot be fully contained in any C_i. Therefore, there exists an arc of this cycle, say (u, v), such that $u \in C_j$ and $v \in C_i$, $j > i$. This violates that $E_{back} \subseteq S$. □

Let (D, k) be the input instance of DEOCT. The algorithm of Theorem 11.0.2 applies the kernelization algorithm of Theorem 4.2.1 to obtain an equivalent instance (D', k'). This is followed by a dynamic programming procedure over the k'-cuts in D' to obtain a γ-vertex sequence of D' of cost at most k.

Proof of Theorem 11.0.2. We will solve DEOCT by doing a dynamic programming over the set $\mathcal{C}$ of k-cuts. Let (D, k) be the input instance. Apply the kernelization algorithm of Theorem 4.2.1 to obtain an equivalent instance (D', k') where the number of vertices in D' is $k^{f(\alpha)}$. Since (D', k') is equivalent to (D, k), it is enough to solve the problem on (D', k'). For ease of notation, we will denote (D', k') by (D, k).

From Lemma 10.1.1, the number of k-cuts in D is at most $\eta = 2^{c'\sqrt{k}\log k} \cdot (k^{f(\alpha)} + 1)^{2\alpha\lceil\sqrt{k}\rceil} \cdot \log k$, where c' is a fixed absolute constant. From Lemma 10.5.1, all these k-cuts can be enumerated in $\eta \cdot k^{\mathcal{O}(1)}$ time. Let us denote by $\mathcal{C}$, the set of k-cuts of D.

Let T denote the dynamic programming table indexed by cuts in and integers $\{0, \ldots, k\}$. For any k-cut $(L, R) \in \mathcal{C}$ and $i \in \{0, \ldots, k\}$, we $T((L, R), i)$ is defined as follows.

$$T((L,R),i) = \begin{cases} 1 & \text{if there exists a } \gamma\text{-vertex sequence } (C_1, \ldots, C_\ell) \text{ of } D[L] \\ & \text{of cost at most } i, \text{ and } (L \setminus \{C_\ell\}, R \cup \{C_\ell\}) \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$$

Note that $T((V(D), \emptyset), k) = 1$ if and only if D has a *deoct* of size at most k. This follows from Lemma 11.2.1.

We now describe how we compute $T((L, R), i)$, for any $(L, R) \in \mathcal{C}$ and $i \in [k]$. For all $i \in [k]$, $T((\emptyset, V(D)), i) = 1$. For any $(L, R) \in \mathcal{C}$, such that $L \neq \emptyset$, and any $i \in [k]$, $T((L, R), i) = 1$ if and only if the following holds: there exists $C \subseteq L$ such that $(L \setminus C, R \cup C) \in \mathcal{C}$, and $T((L \setminus C, R \cup C), i - j) = 1$ where $j = deoct(C) + |\{(u, v) : u \in C, v \in L \setminus C\}|$. Observe that the above describes a recursive procedure that computes all entries in T in time $2^{c^* f(\alpha)\sqrt{k}\log k}$ where c^* is an absolute constant. In total the running time of our algorithm is $2^{cf(\alpha)\sqrt{k}\log k} \cdot n^{\mathcal{O}(1)}$ where c is an absolute constant.

It only remains to prove the correctness of the above procedure. We now prove that for any $(L, R) \in \mathcal{C}$ and $i \in [k]$, $T((L, R), i) = 1$ if and only if there exists a γ-

vertex sequence $(C_1, \ldots, C_\ell)$ of $D[L]$ of cost at most i, and $(L \setminus \{C_\ell\}, R \cup \{C_\ell\}) \in \mathcal{C}$. We prove this by induction on $|L|$. When $|L| = 0$, this is true because of the base case.

In the forward direction, we will show that if $T((L, R), i) = 1$ then there exists a γ-vertex sequence $(C_1, \ldots, C_\ell)$ of $D[L]$ of cost at most i, and $(L \setminus \{C_\ell\}, R \cup \{C_\ell\}) \in \mathcal{C}$. In the above procedure for computing the table T, we set $T((L, R), i) = 1$ only if there exists $C_\ell \subseteq L$, such that $(L \setminus \{C_\ell\}, R \cup \{C_\ell\}) \in \mathcal{C}$ and $T((L \setminus C, R \cup C), i - j) = 1$ where $j = \mathit{deoct}(C) + |\{(u, v) : u \in C, v \in L \setminus C\}|$. Since $T((L \setminus \{C_\ell\}, R \cup \{C_\ell\}), i - j) = 1$, by the induction hypothesis, $D[L \setminus \{C_\ell\}]$ has a γ-vertex sequence of cost at most $i - j$. Let $(C_1, \ldots, C_{\ell-1})$ be the ordering of $L \setminus \{C_\ell\}$ witnessing this, that is, cost of this ordering is at most $i - j$. Since $\mathit{deoct}(C) + |\{(u, v) : u \in C, v \in L \setminus C\}| = j$, the cost of $(C_1, \ldots, C_\ell)$ is at most i. Thus, the $(C_1, \ldots, C_{\ell-1}, C_\ell)$ is a γ-vertex sequence of $D[L]$ of cost at most i.

In the reverse direction, we will show that if $D[L]$ has a γ-vertex sequence of cost at most i and $(L \setminus \{C_\ell\}, R \cup \{C_\ell\}) \in \mathcal{C}$, then $T((L, R), i) = 1$. Let $(C_1, \ldots, C_\ell)$ be a γ-vertex sequence in $D[L]$ of cost at most i such that $(L \setminus \{C_\ell\}, R \cup \{C_\ell\}) \in \mathcal{C}$. Let $j = \mathit{deoct}(C) + |\{(u, v) : u \in C, v \in L \setminus C\}|$. Then the sequence $(C_1, \ldots, C_{\ell-1})$ is a γ-vertex sequence of $D[L \setminus \{C_\ell\}]$ of cost at most $i - j$. Thus, $T((L \setminus \{C_\ell\}, R \cup \{C_\ell\}), i - j) = 1$. Then it follows that our recursive procedure sets $T((L, R), i) = 1$. This concludes the proof.

Chapter 12

Balanced Judicious Bipartition is FPT

More than twenty years ago, Bollobás and Scott [25] defined the notion of *judicious partitioning problems*. Since then, the family of judicious partitioning problems has been extensively studied in the field of Extremal Combinatorics, as can be evidenced by the abundance of structural results described in surveys such as [29, 191]. This rich realm of problems aims to counterbalance the objectives of classical partitioning problems such as Min Cut, Min Bisection, Max Cut and Max Bisection. While these classical problems focus solely on the minimization/maximization of the number of edges crossing the cut (or alternately, the total number of edges inside the parts of the partition), judicious (bi)partitioning problems ask the natural questions of the minimization/maximization of the number of edges lying inside each part of the partition *simultaneously*. Another significant feature of judicious partitioning problems that also distinguishes them from other classical partitioning problems is that they inherently and naturally encompass several objectives, aiming to minimize (or maximize) the number of edges in several sets *simultaneously*.

In this chapter, we shed light on properties of judicious partitioning problems from the viewpoint of the design of algorithms. Up until now, the study of such problems has essentially been overlooked at the algorithmic front, where one of the underlying reasons for this discrepancy might be that standard machinery does not seem to handle them effectively. Specifically, we focus on the Judicious Bipartition problem, where we seek a bipartition that is "judicious" in the sense that neither side has too many edges that lie entirely inside it, and on the Balanced Judicious Bipartition problem, where we also require that the sizes of the sides

themselves are "balanced" in the sense that the number of vertices in both the parts are almost same. Both of these problems were defined in the work by Bollobás and Scott, and have received notable scientific attention since then. Formally, Balanced Judicious Partition is defined as follows.

Balanced Judicious Bipartition (BJB)	**Parameter:** $k_1 + k_2$
Input: A multigraph G, and integers μ, k_1 and k_2	
Question: Does there exist a partition (V_1, V_2) of $V(G)$ such that $\lvert V_1\rvert = \mu$ and for all $i \in \{1, 2\}$, it holds that $\lvert E(G[V_i])\rvert \leq k_i$?	

We note that in the literature, the term BJB refers to the case where $\mu = \lceil \frac{|V(G)|}{2} \rceil$, and hence it is more restricted then the definition above. By dropping the requirement that $|V_1| = \mu$, we get the Judicious Bipartition (JB) problem. By using new crucial insights into these problems on top of the most advanced machinery in Parameterized Complexity to handle partitioning problems,[1] we are able to resolve the question of the Parameterized Complexity of BJB (and hence also of JB). In particular, we prove the following theorem.

Theorem 12.0.1. BJB *can be solved in time* $2^{k^{\mathcal{O}(1)}} \cdot |V(G)|^{\mathcal{O}(1)}$.

Structural Results. Denote $n = |V(G)|$ and $m = |E(G)|$. To survey several structural results about judicious partitioning problems, we first define the notions of *t-cut* and *max (min) t-judicious partitioning*. Given a partition of $V(G)$ into t parts, a t-cut is the number of edges going across the parts, while a max (min) judicious t-partitioning is the maximum (minimum) number of edges in any of the parts. When $t = 2$, we use the standard terms *bipartite-cut* and *judicious bipartitioning*, respectively. Furthermore, by t-judicious partitioning we mean max t-judicious partitioning. As stated earlier, Bollobás and Scott [25] defined the notion of *judicious partitioning problems* in 1993. In that paper, they showed that for any positive integer t and graph G, we can partition $V(G)$ into t sets, $V_1, \ldots, V_t$, so that $|E(G[V_i])| \leq \frac{t}{t+1}m$ for all $i \in \{1, \ldots, t\}$. Bollobás and Scott also studied this problem on graphs of maximum degree Δ, and showed that there exists a partition of $V(G)$ into t sets $V_1, \ldots, V_t$ so that it simultaneously satisfies an upper bound and a lower bound on the number of edges in each part as well as on edges between every pair of parts. Later, Bollobás and Scott [29] gave several new results concerning the extremal bounds of the k-judicious partitioning problem, leaving open other new

[1] To the best of our knowledge, up until now, this machinery has actually only been proven useful to solve one natural problem which could not have been tackled using earlier tools.

questions concerning the tightness of theirt bounds in general and special cases. In [30] they showed an optimal bound for the number of edges inside a part for the judicious partitioning problem on bounded-degree graphs. These problems have also been studied on general hypergraphs [26], uniform hypergraphs [119], 3-uniform hypergraphs [28] and directed graphs [138].

The special cases of judicious partitioning problems called judicious bipartitioning and balanced judicious bipartitioning problems have also been studied intensively. Bollobás and Scott [27] proved an upper bound on judicious bipartitioning and proved that every graph that achieves the essentially best known lower bound on bipartite-cut, given by Edwards in [84] and [85], also achieves this upper bound for judicious bipartitioning. In fact, they showed that this is exact for complete graphs of odd order, which are the only extremal graphs without isolated vertices. Alon et al. [6] gave a non-trivial connection between the size of a bipartite-cut in a graph and judicious partitioning into two sets. In particular, they showed that if a graph has a bipartite-cut of size at least $\frac{m}{2}+\delta$ where $\delta \leq m/30$, then there exists a bipartition (V_1, V_2) of $V(G)$ such that $|E(G[V_i])| \leq \frac{m}{4} - \frac{\delta}{2} + \frac{10\delta^2}{m} + 3\sqrt{m}$ for $i \in \{1,2\}$. They complemented these results by showing an upper bound on the number of edges in each part when $\delta > m/30$. Bollobás and Scott [31] studied similar relations between t-cuts and t-judicious partitionings for $t \geq 3$. Recently, these results were further refined [204, 151]. Xu et al. [203] and Xu and Yu [205] studied balanced judicious bipartitioning where both parts are of almost equal size (that is, one of the sizes is $\lceil \frac{n}{2} \rceil$). Both of these papers concern the following conjecture of Bollobás and Scott [29]: if G is a graph with minimum degree of at least 2, then $V(G)$ admits a balanced bipartition (V_1, V_2) such that for each $i \in \{1,2\}$, $|E(G[V_i])| \leq \frac{m}{3}$. For further results on judicious partitioning, we refer to the surveys [29, 191].

Algorithmic Results. While classical partitioning problems such as Min Cut, Min Bisection, Max Cut and Max Bisection have been studied extensively algorithmically, the same is not true about judicious partitioning problems. Apart from Min Cut, all the above mentioned partitioning problems are NP-complete. These NP-complete partitioning problems were investigated by all algorithmic paradigms meant for coping with NP-completeness, including approximation algorithms and parameterized complexity. In what follows, we discuss known results related to these problems in the realm of parameterized complexity.

First, note that for every graph G, there always exists a bipartition of the vertex set into two parts (in fact equal parts [116, Corollary 1]) such that at least $m/2$ edges are going across. This immediately implies that Max Cut and Max Bi-

SECTION are FPT when parameterized by the cut size (the number of edges going across the partition). This led Mahajan and Raman [153] to introduce the notion of above-guarantee parameterization. In particular, they showed that one can decide whether a graph has a bipartite-cut of size $\frac{m}{2} + k$ in time $\mathcal{O}(m + n + k4^k)$. However, Edwards [84] showed that every connected graph G has a bipartite-cut of size $\frac{m}{2} + \frac{n-1}{4}$. Thus, a more interesting question asks whether finding a bipartite-cut of size at least $\frac{m}{2} + \frac{n-1}{4} + k$ is FPT. Crowston et al. [64] showed that indeed this is the case as they design an algorithm with running time $\mathcal{O}(8^k n^4)$. Recently, Etscheid and Mnich [89] discovered a kernel with a linear number of vertices (improving upon a kernel by Crowston et al. [63]), and the aforementioned algorithm was sped-up to run in time $\mathcal{O}(8^k m)$ [89]. Gutin and Yeo studied an above-guarantee version of MAX BISECTION [116], proving that finding a balanced bipartition such that it has at least $\frac{m}{2} + k$ edges is FPT (also see [166]).[2] In this context MAX BISECTION, it is also relevant to mention the $(k, n-k)$-MAX CUT, which asks for a bipartite-cut of size at least p where one of the sides is of size exactly k. Parameterized by k, this problems is W[1]-hard [38], but parameterized by p, this problem is solvable in time $\mathcal{O}^*(2^p)$ [190] (this result improved upon algorithms given in [34, 194]).

Until recently, the parameterized complexity of MIN BISECTION was open. Approaches to tackle this problem materialized when the parameterized complexity of ℓ-WAY CUT was resolved. Here, given a graph G and positive integers k and ℓ, the objective is to delete at most k edges from G such that it has at least ℓ components. Kawarabayashi and Thorup [129] showed that this problem is FPT (parameterized by k). Later, Chitnis et al. [55] developed a completely new tool based on this, called *randomized contractions*, to deal with plethora of cut problems. Other cut problems that have been shown to be FPT include the generalization of MIN CUT to MULTIWAY CUT and MULTICUT [50, 156, 160]. Eventually, Cygan et al. [69], combining ideas underlying the algorithms developed for MULTIWAY CUT, MULTICUT, ℓ-WAY CUT and randomized contractions together with a new kind of decomposition, showed MIN BISECTION to be FPT. Finally, let us also mention the min c-judicious partitioning (which is a maximization problem), called c-LOAD COLORING, where given a graph G and a positive integer k, the goal is to decide whether $V(G)$ can be partitioned into c parts so that each part has at least k edges. Barbero et al. [13] showed that this problem is FPT (also see [115]).

Despite the abundance of work described above, the parameterized complexity of JB and BJB has not yet been considered. We fill this gap in our studies by

[2]We refer to surveys [154, 117] for details regarding above-guarantee parameterizations.

showing that both of these problems are FPT. It is noteworthy to remark that one can show that the generalization of Min Bisection to c-Min Bisection, where the objective is to find a partition into c-parts such that each of the parts are of almost the same size and there are at most k edges going across different parts, is FPT [69]. However, such a generalization is not possible for either JB or BJB. Indeed, even the existence of an algorithm with running time $n^{f(k)}$, for any arbitrary function f, would imply a polynomial-time algorithm for 3-Coloring, where k is set to 0.

Our Approach. For the sake of readability, our strategy of presentation of our proof consists of the definition of a series of problems, each more "specialized" (in some sense) than the previous one, where each section shows that to eventually solve BJB, it is sufficient to focus on some such problem rather than the previous one. We start by showing that we can focus on the solution of the case of BJB where the input graph is bipartite at the cost of the addition of annotations. For this purpose, we present a (not complicated) Turing reduction that employs a known algorithm for the Odd Cycle Transversal problem (see Section 12.2). The usefulness of the ability to assume that the input graph is bipartite is a key insight in our approach. In particular, the technical parts of our proof crucially rely on the observation that a connected bipartite graph has only two bipartitions (here, we consider bipartitions as ordered pairs). Keeping this intuition in mind, our next step is to reduce the current annotated problem to one where the input graph is also assumed to be connected (this specific argument relies on a simple application of dynamic programming).

Having at hand an (annotated) problem where the input graph is assumed to be a connected bipartite graph, we proceed to the technical part of our proof, which employs the (heavy) machinery developed by Cygan et al. [69]. While this machinery primarily aims to tackle problems where one seeks small cuts in addition to some size constraint, our problem involves a priori seemingly different type of constraints. Nevertheless, we observe that once we handle a connected graph, the removal of any set of k edges (to deal with the size constraint and annotations) would not break the graph into more than $k + 1$ connected components, and each of these components would clearly be a bipartite graph. Hence, we can view (in some sense) our problem as a cut problem. In practice, the relation between our problem and a cut problem is quite more intricate, and to realize our idea, we crucially rely on the fact that the connected components are bipartite graphs, which allows us to "guess" a binary vector specifying the biparition of their vertex sets in the final

solution. This operation entails the employment of coloring functions (employing $k+1$ colors) and their translation into bipartitions (which at a certain point in this chapter, we would start viewing as colorings employing two colors). Let us remark that the machinery introduced by [69] is the computation of a special type of tree decomposition. Accordingly, our approach would eventually involve the introduction of a specialization of BJB that aims to capture the work to perform when handling a bag of the tree decomposition. The definition of this specific problem is very technical, and hence we defer the description of related intuitive explanations to the appropriate locations in Section 12.4, where we have already set up the required notations to discuss it.

12.1 Some Preliminaries

Bold face lowercase letters are used to denote tuples (vectors). For any tuple $\mathbf{v}$, we let $\mathbf{v}[i]$ denote the ith coordinate of $\mathbf{v}$. Given some condition ψ, we define $[\psi] = 1$ if ψ is true and $[\psi] = 0$ otherwise.

A *bipartite graph* is a graph G such that there exists a bipartition (X, Y) of $V(G)$ where X and Y are independent sets. In this chapter, we treat such bipartitions as *ordered pairs*. That is, if (X, Y) is a bipartition of some bipartite graph G, then (Y, X) is assumed to be a *different* bipartition of the graph G. For *connected* bipartite graphs, we have the following simple yet powerful insight.

Proposition 12.1.1 (Folklore)**.** *Any connected bipartite graph G has exactly 2 bipartitions, (X, Y) and (Y, X).*

Let H be some hypergraph. A *spanning forest* of H is a subset $E' \subseteq E(H)$ of minimum size such that the hypergraph induced on E' has the same components as H.

Unbreakability. A *separation* of a graph G is a pair (X, Y) such that $X, Y \subseteq V(G)$, $X \cup Y = V(G)$ and there is no edge with one endpoint in $X \setminus Y$ and the other in $Y \setminus X$. The *order of a separation* (X, Y) is equal to $|X \cap Y|$.

Definition 12.1.1. *Let G be a graph, $A \subseteq V(G)$, and $q, k \in \mathbb{N}$. The set A is said to be* (q, k)-unbreakable *in G if for* every *separation (X, Y) of G of order at most k, either $|(X \setminus Y) \cap A| \leq q$ or $|(Y \setminus X) \cap A| \leq q$.*

We also define a notion of unbreakability in the context of functions.

Definition 12.1.2. *A function $g : U \to [k]_0$ is called* (q,k)-unbreakable *if there exists $i \in [k]_0$ such that $\sum_{j \in [k]_0 \setminus \{i\}} |g^{-1}(j)| \leq q$.*

Let us now claim that there do no exist "too many" (q,k)-unbreakable functions.

Lemma 12.1.1. *For all $q, k \in \mathbb{N}$, the number of (q,k)-unbreakable functions from a universe U to $[k]_0$ is upper bounded by $\sum_{l=0}^{q} \binom{|U|}{l} \cdot q^k \cdot (k+1)$.*

Proof. Let $g : U \to [k]_0$ be some (q,k)-unbreakable function. By the definition of a (q,k)-unbreakable function, there exists $i \in [k]_0$ such that $\sum_{j \in [k]_0 \setminus i} |g^{-1}(j)| \leq q$. There are $(k+1)$ ways of choosing such an index i, $\sum_{l=0}^{q} \binom{|U|}{l}$ ways of choosing at most q elements that are not mapped to i, and at most q^k ways of partitioning this set of at most q elements into k parts. Thus, the total number of such functions g is upper bounded by $\sum_{l=0}^{q} \binom{|U|}{l} q^k (k+1)$. □

12.2 Solving Balanced Judicious Bipartition

In this section, we prove Theorem 12.0.1 under the assumption that we are given an algorithm for an annotated, yet restricted, variant of BJB. Throughout this section, an instance of BJB is denoted by $\text{BJB}(G, \mu, k_1, k_2)$, and we define $k = k_1 + k_2$. Given a partition (V_1, V_2) that witnesses that an instance $\text{BJB}(G, \mu, k_1, k_2)$ is a Yes instance, we think of the vertices in V_1 as colored 1 and the vertices in V_2 as colored 2; hence, we call such a partition a *witnessing coloring* of $\text{BJB}(G, \mu, k_1, k_2)$. To prove Theorem 12.0.1, we first define the Odd Cycle Transversal problem. Here, given a graph G, a set $S \subseteq V(G)$ is called an *odd cycle transversal* if $G \setminus S$ is a bipartite graph.

Odd Cycle Transversal (OCT) **Parameter:** k
Input: An undirected multigraph G, and an integer k.
Output: An odd cycle transversal of G of size at most k, if it exists, otherwise report NO.

An instance of Odd Cycle Transversal is denoted by $\text{OCT}(G, k)$. We say $\text{OCT}(G, k)$ is a No instance if there is no odd cycle transversal of G of size at most

k. The algorithm given by the result below shall be a central component in the design of our algorithm for BJB.

Proposition 12.2.1 ([143]). Odd Cycle Transversal *can be solved in time* $2.3146^k n^{\mathcal{O}(1)}$.

Apart from OCT, we also need to define an auxiliary problem that we call Annotated Bipartite-BJB (AB-BJB). As we proceed with our proofs, we shall continue defining auxiliary problems, where each problem captures a task more specific and technically more challenging than the previous one. The choice of this structure aims to ease the readability of this chapter. Intuitively, AB-BJB is basically the BJB problem on bipartite graphs, with an extra constraint that demands that certain vertices are assigned a particular color by the witnessing coloring. We remark that the necessity of the reduction to bipartite graphs stems from the fact that we would like to employ Proposition 12.1.1 later. The formal definition of AB-BJB is given below.

Annotated Bipartite-BJB (AB-BJB)	**Parameter:** $k_1 + k_2$
Input: A bipartite multigraph G with bipartition (P, Q), $A, B \subseteq V(G)$ such that $A \cap B = \emptyset$, and integers μ, k_1 and k_2.	
Question: Does there exist a partition (V_1, V_2) of $V(G)$ such that $A \subseteq V_1$, $B \subseteq V_2$, $\lvert V_1\rvert = \mu$ and for $i \in \{1, 2\}$, $\lvert E(G[V_i])\rvert \leq k_i$?	

An instance of AB-BJB is denoted by AB-BJB(G, A, B, μ, k_1, k_2). A partition (V_1, V_2) satisfying the above properties is called a *witnessing coloring* of AB-BJB(G, A, B, μ, k_1, k_2). Furthermore, we need the following theorem, proven later in this chapter.

Theorem 12.2.1. AB-BJB *can be solved in time* $2^{k^{\mathcal{O}(1)}} \cdot n^{\mathcal{O}(1)}$.

Let us now turn to focus on the proof of Theorem 12.0.1.

Proof of Theorem 12.0.1. Given an instance BJB(G, μ, k_1, k_2), call the algorithm given by Proposition 12.2.1 with the instance OCT(G, k) as input (recall that $k = k_1 + k_2$).

Claim 12.2.1. *If* OCT(G, k) *is a* No *instance, then* BJB(G, μ, k_1, k_2) *is a* No *instance.*

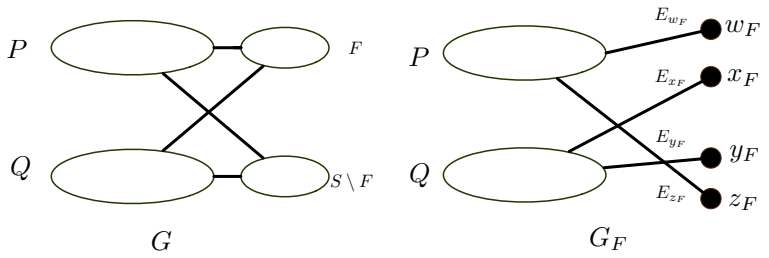


Figure 12.1: The construction in the proof of Theorem 12.0.1.

Proof. Suppose $\mathrm{BJB}(G, \mu, k_1, k_2)$ is a YES instance. Let (V_1, V_2) be a witnessing coloring for this instance. Let $E' = E(G[V_1]) \cup E(G[V_2])$. Then, observe that $G \setminus E'$ is a bipartite graph. Let V' be a set of vertices of minimum size such that every edge in E' has at least one endpoint in V'. Since $|E'| \leq k$, it holds that $|V'| \leq k$. Moreover, $G \setminus V'$ is bipartite. Therefore, V' is an odd cycle transversal of G of size at most k. Thus, $\mathrm{OCT}(G, k)$ is a YESinstance. □

Henceforth, let S be an odd cycle transversal of G of size at most k. Then, $G \setminus S$ is a bipartite graph. Fix some bipartition (P, Q) of $G \setminus S$. Let $\mathcal{F}$ be the family of all subsets of S, that is, $\mathcal{F} = 2^S$. For any $F \in \mathcal{F}$, denote $l_1^F = |E(G[F])|$ and $l_2^F = |E(G[S \setminus F])|$, and let G_F be the graph constructed as follows (see Fig. 12.1).

- $V(G_F) = V(G \setminus S) \cup \{w_F, x_F, y_F, z_F\}$, where w_F, x_F, y_F, z_F are new distinct vertices.
- $E(G_F) = E(G \setminus S) \cup E_{w_F} \cup E_{x_F} \cup E_{y_F} \cup E_{z_F}$, where the *multisets* $E_{w_F}, E_{x_F}, E_{y_F}$ and E_{z_F} are defined as follows.
  - for each edge $(u, v) \in E(G)$, such that $u \in F$ and $v \in P$, there is an edge $(w_F, v) \in E_{w_F}$,
  - for each edge $(u, v) \in E(G)$, such that $u \in F$ and $v \in Q$, there is an edge $(x_F, v) \in E_{x_F}$,
  - for each edge $(u, v) \in E(G)$, such that $u \in S$ and $v \in Q$, there is an edge $(y_F, v) \in E_{y_F}$,
  - for each edge $(u, v) \in E(G)$, such that $u \in S$ and $v \in P$, there is an edge $(z_F, v) \in E_{z_F}$.

Observe that G_F is a bipartite graph with $(P \cup \{x_F, y_F\}, Q \cup \{w_F, z_F\})$ as a bipartition.

Claim 12.2.2. BJB(G, μ, k_1, k_2) *is a* YES *instance if and only if there exists* $F \in \mathcal{F}$ *such that* AB-BJB$(G_F, \{w_F, x_F\}, \{y_F, z_F\}, \mu - |F| + 2, k_1 - l_1^F, k_2 - l_2^F)$ *is a* YES*-instance.*

Proof. In the forward direction, suppose that BJB(G, μ, k_1, k_2) is a YES instance, and let (V_1, V_2) be a witnessing coloring for BJB(G, μ, k_1, k_2). Moreover, let $F = V_1 \cap S$. Now, we define a partition (V_1', V_2') of $V(G_F)$ as follows: $V_1' = (V_1 \setminus S) \cup \{w_F, x_F\}$ and $V_2' = (V_2 \setminus S) \cup \{y_F, z_F\}$. Let us now argue that (V_1', V_2') is a witnessing coloring for AB-BJB$(G_F, \{w_F, x_F\}, \{y_F, z_F\}, \mu - |F| + 2, k_1 - l_1^F, k_2 - l_2^F)$. First, by the construction of (V_1', V_2'), we have that $\{w_F, x_F\} \subseteq V_1'$ and $\{y_F, z_F\} \subseteq V_2'$. Second, as $V_1' = (V_1 \setminus S) \cup \{w_F, x_F\}$, we also have that $|V_1'| = |V_1| - |F| + 2 = \mu + |F| + 2$. Third, observe that for any $|E(G[V_1'])| = |E(G[V_1])| - |E(G[F])|$ and $|E(G[V_2'])| = |E(G[V_2])| - |E(G[S \setminus F])|$. Thus, for $i \in [2]$, $|E(G[V_i])| \leq k_i - l_i^F$.

In the backward direction, suppose that there exists an $F \in \mathcal{F}$ such that AB-BJB$(G_F, \{w_F,\ x_F\}, \{y_F, z_F\}, \mu - |F| + 2, k_1 - l_1^F, k_2 - l_2^F)$ is a YES instance, and let (V_1', V_2') be a witnessing coloring for this instance. We now define a partition (V_1, V_2) of $V(G)$ as follows: $V_1 = (V_1' \cap V(G)) \cup F$ and $V_2 = (V_2' \cap V(G)) \cup (S \setminus F)$. Let us now argue that (V_1, V_2) is a witnessing coloring for BJB(G, μ, k_1, k_2). From the definition of V_1, and since $V(G) = (V(G_F) \setminus \{w_F, x_F, y_F, z_F\}) \cup S$ and $S \cap V(G_F) = \emptyset$, we have that $|V_1| = |V_1'| - |\{x_F, y_F\}| + |F| = \mu - |F| + 2 - 2 + |F| = \mu$. Moreover, observe that $|E(G[V_1])| = |E(G[V_1'])| + |E(G[F])| \leq k_1 + l_1^F$ and $|E(G[V_2])| = |E(G[V_2'])| + |E(G[S \setminus F])| \leq k_2 + l_2^F$. This concludes the proof of the claim. □

Thus, to solve an instance of BJB, it is enough to solve $2^{|S|} \leq 2^k$ instances of AB-BJB. Hence, by Theorem 12.2.1, BJB can be solved in time $2^{k^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$. □

12.3 Solving ANNOTATED BIPARTITE-BJB

Recall the problem definition of ANNOTATED BIPARTITE-BJB (AB-BJB) from Section 12.2. In this section, we prove Theorem 12.2.1. For this purpose, let us define another auxiliary problem, which we call ANNOTATED BIPARTITE CONNECTED-BJB (ABC-BJB). Intuitively, ABC-BJB is exactly the same problem as AB-BJB where we are interested in an answer for every choice of $\mu \in [n]_0$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, and additionally we demand the input graph to be connected.

ANNOTATED BIPARTITE CONNECTED-BJB (ABC-BJB) **Parameter:** $k_1 + k_2$
Input: A connected bipartite multigraph $G = (P, Q)$, $A, B \subseteq V(G)$ such that $A \cap B = \emptyset$, and integers k_1 and k_2.
Output: For all $\mu \in [n]_0$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, output a binary value, $\texttt{aJP}[\mu, l_1, l_2]$, which is 1 if and only if there exists a partition (V_1, V_2) of $V(G)$ such that

- $A \subseteq V_1$ and $B \subseteq V_2$,
- $|V_1| = \mu$, and
- for $i \in \{1, 2\}$, $|E(G[V_i])| \leq l_i$.

For any $\mu \in [n]_0$, $l_1 \in [k_1]_0$, $l_2 \in [k_2]_0$, a partition witnessing that $\texttt{aJP}[\mu, l_1, l_2] = 1$ is called a *witnessing coloring* for $\texttt{aJP}[\mu, l_1, l_2] = 1$. Moreover, an instance of ABC-BJB is denoted by ABC-BJB(G, A, B, k_1, k_2). In the rest of this chapter, we prove the following theorem.

Theorem 12.3.1. ABC-BJB *can be solved in time* $2^{k^{\mathcal{O}(1)}} \cdot n^{\mathcal{O}(1)}$.

Having Theorem 12.3.1 at hand, a simple application of the method of dynamic programming results in the proof of Theorem 12.2.1.

Proof of Theorem 12.2.1. Let AB-BJB(G, A, B, μ, k_1, k_2) be an instance of AB-BJB. Let $C_1, \ldots, C_r$ be the connected components of G. For all $i \in [r]$, let $A_i = A \cap C_i$ and $B_i = B \cap C_i$. Let $I_i =$ ABC-BJB$(C_i, A_i, B_i, k_1, k_2)$. Let $\texttt{aJP}_i$ be the output table for the instance I_i, returned by the algorithm of Theorem 12.3.1. For any $j \in [r]$, let $G_j = G[\bigcup_{i \in [j]} C_i]$. Note that $G = G_r$. Let us define a 4-dimensional binary table $\texttt{M}$ in the following way. For all $i \in [r]$, $\mu' \in [|V(G)|]_0$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, $\texttt{M}[i, \mu', l_1, l_2] = 1$ if and only if there exists a partition (V_1, V_2) of $V(G_i)$ such that $(A \cap G_i) \subseteq V_1$, $(B \cap G_i) \subseteq V_2$, $|V_1| = \mu'$ and for $j \in \{1, 2\}$, $|E(G[V_j])| \leq l_j$. Observe that AB-BJB(G, A, B, μ, k_1, k_2) is a YES-instance if and only if $\texttt{M}[r, \mu, k_1, k_2] = 1$. We now compute $\texttt{M}[r, \mu, k_1, k_2]$ recursively using the following recurrences.

$$\texttt{M}[1, \mu', l_1, l_2] = \texttt{aJP}_1(\mu', l_1, l_2)$$

For all $i \in \{2, \ldots, r\}$, $\mu' \in [|V(G)|]_0$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$,

$$\mathtt{M}[i, \mu', l_1, l_2] = \bigvee_{\substack{\mu'=\mu^1+\mu^2 \\ l_1=l_1^1+l_1^2 \\ l_2=l_2^1+l_2^2}} (\mathtt{M}[i-1, \mu^1, l_1^1, l_2^1] \wedge \mathtt{aJP}_i[\mu^2, l_1^2, l_2^2]),$$

where for all $j \in \{1, 2\}$, μ^j, l_1^j and l_2^j are non-negative integers.

Note that the time taken to compute $\mathtt{M}[r, \mu, k_1, k_2]$ is at most $(r \cdot n^2 \cdot k_1^2 \cdot k_2^2 \cdot \tau)$, where τ is the time taken to solve an instance of ABC-BJB. Since from Theorem 12.3.1, an instance of ABC-BJB can be solved in time $2^{k^{\mathcal{O}(1)}} \cdot n^{\mathcal{O}(1)}$ and $r \leq n$, AB-BJB can be solved in time $2^{k^{\mathcal{O}(1)}} \cdot n^{\mathcal{O}(1)}$. □

12.4 Solving Annotated Bipartite Connected-BJB

Recall the problem definition of ABC-BJB from Section 12.3. In this section, we prove Theorem 12.3.1. Let us start by stating a known result that is a crucial component of our proof. By this result, we would have an algorithm that efficiently computes a special type of tree decomposition, that we call a *highly connected tree decomposition*, where every bag is "highly-connected" rather than "small" as in the case of standard tree decompositions. While this property is the main feature of this decomposition, it is also equipped with other beneficial properties, such as a (non-trivial) upper bound on the size of its adhesions, which are *all* exploited by our algorithm.

Theorem 12.4.1 ([69]). *There exists an $2^{\mathcal{O}(k^2)}n^2m$-time algorithm that, given a* connected *graph G together with an integer k, computes a tree decomposition (T, β) of G with at most n nodes such that the following conditions hold, where $\eta = 2^{\mathcal{O}(k)}$.*

1. *For each $t \in V(T)$, the graph $G[\gamma(t) \setminus \sigma(t)]$ is connected and $N(\gamma(t) \setminus \sigma(t)) = \sigma(t)$.*

2. *For each $t \in V(T)$, the set $\beta(t)$ is (η, k)-unbreakable in $G[\gamma(t)]$.*

3. *For each non-root $t \in V(T)$, we have that $|\sigma(t)| \leq \eta$ and $\sigma(t)$ is $(2k, k)$-unbreakable in $G[\gamma(\text{parent}(t))]$.*

In order to process such a tree decomposition in a bottom-up fashion, relying on the method of dynamic programming, we need to address a specific problem associated with every bag, called Hypergraph Painting (HP). We chose the name HP to be consistent with the choice of problem name in [69], yet we stress that our problem is more general than the one in [69] (since the handling of a bag in our case is more intricate than the one in [69]).

Roughly speaking, an input of HP would consist of the following components. First, we are given "budget" parameters k_1 and k_2 as in an instance of ABC-BJB. Second, we are given an argument b which would simply be n (to upper bound $|\gamma(t)|$) when we construct an instance of HP while processing some node t in the tree decomposition. Third, we are given a hypergraph H which would essentially be the graph $G[\beta(t)]$ to which we add hyperedges. Each hyperedge F of H is supposed to represent the sets $\sigma(\widehat{t})$ for each child $\widehat{t}$ of t. Fourth, we are given an integer q whose purpose is clarified in the discussion below the definition of HP (in Definition 12.4.3). Finally, for every hyperedge F, we are given a function $f_F : [k]_0^F \times [b]_0 \times [k_1]_0 \times [k_2]_0 \rightarrow \{0,1\}$. To roughly understand the meaning of this function, first recall that F is supposed to represent $\sigma(\widehat{t})$ for some child $\widehat{t}$ of t. Now, the function f_F aims to capture all information obtained while we processed the child $\widehat{t}$ of t that might be relevant to the node t. In particular, let us give an informal, intuitive interpretation of an element (Γ, μ, l_1, l_2) in the domain of f_F. For this purpose, note that when we remove at most k edges from the (connected) graph $G[\gamma(\widehat{t})]$, we obtain at most $k+1$ connected components. The function Γ can be thought of as a method to assign to each vertex in $\sigma(\widehat{t})$ the connected component in which it should lie. Such information is extremely useful since each such connected component is in particular a bipartite graph, and hence by relying on Proposition 12.1.1 and an exhaustive search, we would be able to use it to extract a witnessing coloring for an instance of ABC-BJB. The arguments μ, l_1 and l_2 can be thought of as those in the definition of an output of ABC-BJB. Now, the value $f_F(\Gamma, \mu, l_1, l_2)$ aims to indicate whether Γ, μ, l_1 and l_2 are "realizable" in the context of the child $\widehat{t}$ (the precise meaning of this value would become clearer later, once we establish additional necessary definitions).

Let us now give the formal definition of HP. In this definition, we denote $k = k_1 + k_2$.

HYPERGRAPH PAINTING (HP)

Input: Integers k_1, k_2, b, d and q, a multi-hypergraph H with hyperedges of size at most d, and for all $F \in E(H)$, a function $f_F : [k]_0^F \times [b]_0 \times [k_1]_0 \times [k_2]_0 \to \{0,1\}$.

Question: For all $0 \leq \mu \leq b$, $0 \leq l_1 \leq k_1$, $0 \leq l_2 \leq k_2$, output the binary value

$$\texttt{aHP}[\mu, l_1, l_2] = \bigvee_{\Upsilon : V(H) \to [k]_0} \bigvee_{\substack{\{\mu^F\}|_{F \in E(H)} \\ \{l_1^F\}|_{F \in E(H)} \\ \{l_2^F\}|_{F \in E(H)}}} \bigwedge_{F \in E(H)} f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F),$$

where $\sum_{F \in E(H)} \mu^F = \mu$, $\sum_{F \in E(H)} l_1^F \leq l_1$, $\sum_{F \in E(H)} l_2^F \leq l_2$ and each of μ^F, l_1^F and l_2^F is a non-negative integer.

For a particular choice of μ, l_1 and l_2, a function Υ witnessing that $\texttt{aHP}[\mu, l_1, l_2] = 1$ is called a witnessing coloring for $\texttt{aHP}[\mu, l_1, l_2]$. An instance of HYPERGRAPH PAINTING is denoted by $\mathrm{HP}(k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$. Observe that, q is part of the input to an instance of HP, but does not appear in the problem definition. The reason for putting q in the input will become clear when we define *favorable* instances of HP. These are the instances that will be of interest to us throughout this article. Although we are not able to tackle HP efficiently at its full generality, we are still able to solve those instances that are constructed when we would like to "handle" a single bag in a highly connected tree decomposition. Such instances are formalized as favorable instances. For the sake of clarity, let us now address the beneficial properties that these instances satisfy individually, where each of them ultimately aims to ease our search for a witnessing coloring. The first property, called local unbreakability, unconditionally restricts the way a function $\Gamma : F \to [k]_0$, to be thought of as a restriction of the witnessing coloring we seek, can color a hyperedge F so that the value of f_F is 1.[3]

Definition 12.4.1 (Local Unbreakability)**.** *An instance*

$$\mathrm{HP}(k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$$

is locally unbreakable *if every $F \in E(H)$ satisfies the following property: for any $\Gamma : F \to [k]_0$ that is not $(3k^2, k)$-unbreakable, $f_F(\Gamma, \mu, l_1, l_2) = 0$ for all $0 \leq \mu \leq b$, $0 \leq l_1 \leq k_1$ and $0 \leq l_2 \leq k_2$.*

The second property, called connectivity, implies that if we would like to use a function $\Gamma : F \to [k]_0$ to color a hyperedge (as a restriction of a witnessing coloring)

[3]In this context, it may be insightful to recall Lemma 12.1.1.

with more than one color, then we would have to "pay" at least 1 from our budget $l_1 + l_2$.

Definition 12.4.2 (Connectivity)**.** *An instance* $\mathrm{HP}(k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$ *is* connected *if every* $F \in E(H)$ *satisfies the following property: for any* $\Gamma : F \to [k]_0$ *for which there exist distinct* $i, j \in [k]_0$ *such that* $|\Gamma^{-1}(i)|, |\Gamma^{-1}(j)| > 0$*, it holds that* $f_F(\Gamma, \mu, l_1, l_2) = 1$ *only if* $l_1 + l_2 \geq 1$*.*

The third property, called global unbreakability, directly restricts our "solution space" by implying that we only need to determine whether there exists a (q, k)-unbreakable witnessing coloring.

Definition 12.4.3 (Global Unbreakability)**.** *An instance*

$$\mathrm{HP}(k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$$

is globally unbreakable *if for all* $0 \leq \mu \leq b$*,* $0 \leq l_1 \leq k_1$*,* $0 \leq l_2 \leq k_2$*: if* $\mathtt{aHP}[\mu, l_1, l_2] = 1$*, then there exists a witnessing coloring* $\Upsilon : V(H) \to [k]_0$ *that is* (q, k)*-unbreakable.*

An instance $HP(k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$ is called a *favorable instance of* HP if it is locally unbreakable, connected and globally unbreakable. For such instances we have the following theorem.

Theorem 12.4.2. HP *on favorable instances is solvable in time*

$$2^{\mathcal{O}(\min(k,q)\log(k+q))} d^{\mathcal{O}(k^3)} |E(H)|^{\mathcal{O}(1)}.$$

The proof of this theorem is very technical, involving non-trivial analysis of a very "messy" picture obtained by guessing part of a hypothetical witnessing coloring via the method of color coding. We defer the proof of Theorem 12.4.2 to Section 12.5.

From now onwards, to simplify the presentation of arguments ahead with respect to ABC-BJB, we would abuse notation and directly define a witnessing coloring as a function rather than a partition. More precisely, the term witnessing coloring for $\mathtt{aJP}[\mu, l_1, l_2] = 1$ would refer to a function $col : V(G) \to \{V_1, V_2\}$ such that $A \subseteq V_1$, $B \subseteq V_2$, $|V_1| = \mu$ and for $i \in \{1, 2\}$, $|E(G[V_i])| \leq l_i$. To proceed to our proof of Theorem 12.3.1, we first need to introduce an additional notation. Roughly speaking, this notation translates a coloring Υ of the form that witnesses some $\mathtt{aHP}[\mu, l_1, l_2] = 1$ to a coloring of the form that witnesses $\mathtt{aJP}[\mu, l_1, l_2] = 1$ via some tuple $\mathbf{v} \in \{0, 1\}^{k+1}$. Formally,

Definition 12.4.4. *For a tuple $\mathbf{v} \in \{0,1\}^{k+1}$, bipartite graph G with bipartition (P,Q), $X \subseteq V(G)$ and $\Upsilon : X \to [k]_0$, define $\widehat{\Upsilon}_{\mathbf{v}} : X \to \{V_1, V_2\}$ as follows.*

- *For all $v \in P \cap X$, $\widehat{\Upsilon}_{\mathbf{v}}(v) = V_1$ if and only if $\mathbf{v}[\Upsilon(v)] = 0$.*
- *For all $v \in Q \cap X$, $\widehat{\Upsilon}_{\mathbf{v}}(v) = V_1$ if and only if $\mathbf{v}[\Upsilon(v)] = 1$.*

Suppose we are given an instance ABC-BJB(G, A, B, k_1, k_2). Fix some bipartition (P,Q) of G. Let (T, β) be the highly connected tree decomposition computed by the algorithm of Theorem 12.4.1, and let r be the root of T. In what follows, $\eta = 2^{\mathcal{O}(k)}$ as in Theorem 12.4.1, and $q = (\eta + k)k$. We now proceed to define a binary variable that is supposed to represent the answer we would like to compute when we process the bag of a specific node of the tree. Hence, one of the arguments is a node t, and three additional arguments are $\mu \in [n]_0, l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$. However, we cannot be satisfied with one answer, but need an answer for every possible "interaction" between the bag of t and the bag of its parent t'. Thus, the definition also includes a coloring of $\sigma(t)$. The tuple $\mathbf{v} \in \{0,1\}^{k+1}$ is necessary for the translation process described in Definition 12.4.4 (the way in which we shall obtain such a "right" tuple later in the proof would essentially rely on brute-force).

Definition 12.4.5. *Given $t \in V(T)$, a $(3k^2, k)$-unbreakable function $\Upsilon^\sigma : \sigma(t) \to [k]_0$, a tuple $\mathbf{v} \in \{0,1\}^{k+1}$, and integers $\mu \in [n]_0, l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, the binary variable $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$ is 1 if and only if there exists $\Upsilon : \gamma(t) \to [k]_0$ extending Υ^σ such that*

1. *The translation $\widehat{\Upsilon}_{\mathbf{v}}$ maps to V_1 exactly μ vertices, that is, $|\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)| = \mu$.*
2. *The translation $\widehat{\Upsilon}_{\mathbf{v}}$ maps $A \cap \gamma(t)$ to V_1 and $B \cap \gamma(t)$ to V_2, that is, $A \cap \gamma(t) \subseteq \widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)$ and $B \cap \gamma(t) \subseteq \widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)$.*
3. *For all $i \in \{1,2\}$, it holds that $|E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_i)])| \leq l_i$.*
4. *The set of edges between vertices receiving different colors by Υ is exactly the set of edges between vertices that are mapped to the same side by the translation $\widehat{\Upsilon}_{\mathbf{v}}$, that is,*

$$\bigcup_{i,j \in [k]_0, i \neq j} E(\Upsilon^{-1}(i), \Upsilon^{-1}(j)) = E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)]) \cup E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)]).$$

A function Υ as above is called a *witnessing coloring* for $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$. Recall r refers to the root of the tree decomposition (T, β).

Lemma 12.4.1. *For any $\mu \in [n]_0$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, $\mathtt{aJP}[\mu, l_1, l_2] = 1$ if and only if there exists $\mathbf{v} \in \{0,1\}^{k+1}$ such that $y[r, \emptyset, \mathbf{v}, \mu, l_1, l_2] = 1$.*

Proof. Let us prove the backward direction first. Let $\mathbf{v} \in \{0,1\}^{k+1}$ be such that $y[r, \emptyset, \mathbf{v}, \mu, l_1, l_2] = 1$ and let $\Upsilon : V(G) \to [k]_0$ be one of its witnessing coloring. Then, Definition 12.4.5 directly implies that $\widehat{\Upsilon}_{\mathbf{v}}$ is a witnessing coloring for $\mathtt{aJP}[\mu, l_1, l_2] = 1$.

For the forward direction, let $col : V(G) \to \{V_1, V_2\}$ be a witnessing coloring for $\mathtt{aJP}[\mu, l_1, l_2]$. Let $X = E(G[col^{-1}(V_1)]) \cup E(G[col^{-1}(V_2)])$. Let $C_0, \ldots, C_s$ be the connected components of $G \setminus X$. Since $X \subseteq E(G)$ and $|X| \leq l_1 + l_2 \leq k_1 + k_2 = k$, $G \setminus X$ has at most $k+1$ connected components, therefore $s \leq k$. For any $i \in [s]_0$, let $(P_i = (P \cap C_i), Q_i = (Q \cap C_i))$ be a bipartition of C_i (recall that G is a connected bipartite graph with fixed bipartition (P, Q)).

Claim 12.4.1. *For any $i \in [s]_0$, either both $P_i \subseteq col^{-1}(V_1)$ and $Q_i \subseteq col^{-1}(V_2)$ or both $P_i \subseteq col^{-1}(V_2)$ and $Q_i \subseteq col^{-1}(V_1)$.*

Proof. Consider a bipartition (P'_i, Q'_i) of C_i, where $P'_i = col^{-1}(V_1) \cap C_i$ and $Q'_i = col^{-1}(V_2) \cap C_i$. Since C_i is connected, from Proposition 12.1.1, either $P_i = P'_i$ and $Q_i = Q'_i$, or $P_i = Q'_i$ and $Q_i = P'_i$. Hence the claim follows. □

Let us now construct a k-length binary string, $\mathbf{v}$, as follows. For any $i \in [s]_0$, $\mathbf{v}[i] = 0$ if and only if $P_i \subseteq col^{-1}(V_1)$ and $Q_i \subseteq col^{-1}(V_2)$. For $i \in \{s+1, \ldots, k\}$, $\mathbf{v}[i] = 0$.

Define $\Upsilon : V(G) \to [k]_0$ as follows. For any $v \in V(G)$, $\Upsilon(v) = i$ if and only if $v \in C_i$.

Claim 12.4.2. $\widehat{\Upsilon}_{\mathbf{v}} = col$.

Proof. Consider some vertex $v \in V(G)$. Denote $V_j = col(v)$, $i = \Upsilon(v)$ and $b = \mathbf{v}[i]$, and note that $j \in \{1,2\}$, $i \in [k]_0$ and $b \in \{0,1\}$. We divide the argument into two cases corresponding to whether $v \in P_i$ or $v \in Q_i$. Since $v \in col^{-1}(V_j)$, if $v \in P_i$, then by Claim 12.4.1, $P_i \subseteq col^{-1}(V_j)$ and $Q_i \subseteq col^{-1}(V_{3-j})$. Thus, by the construction of $\mathbf{v}$, $b = j - 1$. Hence, by the definition of $\widehat{\Upsilon}_{\mathbf{v}}$, $\widehat{\Upsilon}_{\mathbf{v}}(v) = V_j$. Similarly, if $v \in Q_i$, then by Claim 12.4.1, $Q_i \subseteq col^{-1}(V_j)$ and $P_i \subseteq col^{-1}(V_{3-j})$. Thus, by the construction of $\mathbf{v}$, $b = 2 - j$. Hence, by the definition of $\widehat{\Upsilon}_{\mathbf{v}}$, $\widehat{\Upsilon}_{\mathbf{v}}(v) = V_j$.

Since the choice of v was arbitrary, by the definition of $\widehat{\Upsilon}_{\mathbf{v}}$, we have that $\widehat{\Upsilon}_{\mathbf{v}}(v) = V_j$. □

Claim 12.4.3. *For the binary string* $\mathbf{v}$ *constructed as above, the function* Υ *constructed above is a witnessing coloring for* $y[r, \emptyset, \mathbf{v}, \mu, l_1, l_2] = 1$.

Proof. Since $\widehat{\Upsilon}_{\mathbf{v}} = col$, from the definition of col, we have that $|\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)| = \mu$, $A \subseteq \widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)$, $B \subseteq \widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)$, and for all $i \in \{1, 2\}$, $|E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_i)])| \leq l_i$. Observe that $\bigcup_{i,j \in [k]_0, i \neq j} E(\Upsilon^{-1}(i), \Upsilon^{-1}(j)) = X$. Therefore, $\bigcup_{i,j \in [k]_0, i \neq j} E(\Upsilon^{-1}(i), \Upsilon^{-1}(j)) = E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)]) \cup E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)])$. Thus, Υ is a witnessing coloring for

$$y[r, \emptyset, \mathbf{v}, \mu, l_1, l_2] = 1$$

. □

This concludes the proof of the lemma. □

By Lemma 12.4.1, it is sufficient to compute $y[r, \emptyset, \mathbf{v}, \mu, l_1, l_2]$ for all $\mu \in [n]$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$. To this end, we need to compute $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$ for every node $t \in V(T)$, function $\Upsilon^\sigma : \sigma(t) \to [k]_0$ that is $(3k^2, k)$-unbreakable, tuple $\mathbf{v} \in \{0, 1\}^{k+1}$, and integers $\mu \in [n]_0, l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$. Here, we employ bottom-up dynamic programming over the tree decomposition (T, β). Let us now zoom into the computation of $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$ for all $\mu \in [n]$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, for some specific t, Υ^σ and $\mathbf{v}$. Note that we now assume that values corresponding to the children of t (if such children exist) have been already computed correctly. Moreover, note that $|\sigma(t)| \leq \eta$, the number of $(3k^2, k)$-unbreakable functions $\Upsilon^\sigma : \sigma(t) \to [k]_0$ is at most $|\eta|^{k^{\mathcal{O}(1)}} = 2^{k^{\mathcal{O}(1)}}$ (by Lemma 12.1.1), and the number of binary vectors of size $k + 1$ is at most 2^{k+1}. Thus, the total running time would consist of the computation time of (T, β), and at most $2^{k^{\mathcal{O}(1)}} \cdot n^2$ times the computation time for a set of values as the one we examine now. Hence, it remains to show how to compute the current set of values in time $2^{k^{\mathcal{O}(1)}} \cdot n^{\mathcal{O}(1)}$.

To compute our current set of values, let us construct an instance

$$\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F \in E(H)})$$

of HP where $V(H) = \beta(t)$, and $E(H)$ and $\{f_F\}|_{F \in E(H)}$ are defined as follows.

1. **Type-1 Hyperedges.** For all $v \in \beta(t)$, insert $F = \{v\}$ into $E(H)$. Define

$f_F : [k]_0^F \times [n]_0 \times [k_1]_0 \times [k_2]_0 \to \{0,1\}$ as

$$f_F(\Gamma, \mu, l_1, l_2) = \begin{cases} 0, & \text{if } v \in \sigma(t) \text{ and } \Gamma(v) \neq \Upsilon^{\sigma}(v) \\ 1, & \text{if } v \in A,\ \widehat{\Gamma}_{\mathbf{v}}(F) = V_1,\ l_1 = l_2 = 0 \text{ and } \mu = 1 \\ 1, & \text{if } v \in B,\ \widehat{\Gamma}_{\mathbf{v}}(F) = V_2,\ l_1 = l_2 = 0 \text{ and } \mu = 0 \\ 1, & \text{if } v \notin A \cup B,\ l_1 = l_2 = 0 \text{ and } \mu = [\widehat{\Gamma}_{\mathbf{v}}(F) = V_1] \\ 0, & \text{otherwise} \end{cases}$$

Informally speaking, we introduce this kind of hyperedges to account for the number of vertices in $\beta(t)$ that go to V_1 (and hence contribute to μ).

2. **Type-2 Hyperedges.** For all $(u,v) \in E(G[\beta(t)])$, add $F = \{u,v\}$ in $E(H)$. Define $f_F : [k]_0^F \times [n]_0 \times [k_1]_0 \times [k_2]_0 \to \{0,1\}$ as

$$f_F(\Gamma, \mu, l_1, l_2) = \begin{cases} 0, & \text{if } \mu \neq 0 \\ 1, & \text{if } \widehat{\Gamma}_{\mathbf{v}}(u) \neq \widehat{\Gamma}_{\mathbf{v}}(v) \text{ and } \Gamma(u) = \Gamma(v) \\ 1, & \text{if } \widehat{\Gamma}_{\mathbf{v}}(u) = \widehat{\Gamma}_{\mathbf{v}}(v) = V_1, l_1 \geq 1 \text{ and } \Gamma(u) \neq \Gamma(v) \\ 1, & \text{if } \widehat{\Gamma}_{\mathbf{v}}(u) = \widehat{\Gamma}_{\mathbf{v}}(v) = V_2, l_2 \geq 1 \text{ and } \Gamma(u) \neq \Gamma(v) \\ 0, & \text{otherwise} \end{cases}$$

We introduce this kind of hyperedges to account for the number of edges in $G[\beta(t)]$ that contribute towards the budget k_1 and k_2.

3. **Type-3 Hyperedges.** For all $\widehat{t} \in V(T)$ that is a child of t in the tree T, insert $F = \sigma(\widehat{t})$ into $E(H)$. Define $f_F : [k]_0^F \times [n]_0 \times [k_1]_0 \times [k_2]_0 \to \{0,1\}$ as

$$f_F(\Gamma, \mu, l_1, l_2) = \begin{cases} 0, & \text{if } \Gamma \text{ is not } (3k^2, k)\text{-unbreakable or} \\ & \quad y[\widehat{t}, \Gamma, \mathbf{v}, \mu + \mu', l_1 + l_1', l_2 + l_2'] = 0 \\ 1, & \text{otherwise} \end{cases}$$

where $\mu' = |\widehat{\Gamma}_{\mathbf{v}}^{-1}(V_1)|$, and $l_i' = |\{\{u,v\} \in E(G[\sigma(\widehat{t})]) : \widehat{\Gamma}_{\mathbf{v}}(u) = \widehat{\Gamma}_{\mathbf{v}}(v) = V_i\}|$ for $i \in [2]$.

This kind of hyperedges encapsulate the partial partitions of the graphs induced by $\gamma(\widehat{t})$, where $\widehat{t}$ is a child of t.

Let us first claim that witnessing colorings related to $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F \in E(H)})$ are useful in the sense that they can be extended to witnessing colorings

for the binary values in which we are interested.

Lemma 12.4.2. *For all $\mu \in [n]$, $l_1 \in [k_1]_0$, and $l_2 \in [k_2]_0$, if $\mathtt{aHP}[\mu, l_1, l_2] = 1$, then $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$. In fact, for any witness $\Upsilon : \beta(t) \to [k]_0$ of $\mathtt{aHP}[\mu, l_1, l_2] = 1$, there exists a function $\Upsilon' : \gamma(t) \to [k]_0$ that extends Υ and witnesses $y[t, \mathbf{v}, \Upsilon^\sigma, \mu, l_1, l_2] = 1$.*

Proof. If $\mathtt{aHP}[\mu, l_1, l_2] = 1$, let $\Upsilon : \beta(t) \to [k]_0$ be a witnessing coloring for $\mathtt{aHP}[\mu, l_1, l_2] = 1$. Then, there exist $\sum_{F \in E(H)} \mu^F = \mu$, $\sum_{F \in E(H)} l_1^F \leq l_1$ and $\sum_{F \in E(H)} l_2^F \leq l_2$, such that for all $F \in E(H)$, $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F) = 1$. In particular, the following holds.

1. Since for any *type-1* hyperedge F, it holds that $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F) = 1$, we overall have that $\Upsilon^\sigma \subseteq \Upsilon$, $A \cap \beta(t) \subseteq \widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)$, $B \cap \beta(t) \subseteq \widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)$ and

$$\sum_{F \text{ is a type-1 hyperedge}} \mu^F = |\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1) \cap \beta(t)|.$$

2. Since for any *type-2* hyperedge F and $i \in \{1, 2\}$, it holds that $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F) = 1$, we overall have that

$$|E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_i) \cap \beta(t)])| \leq \sum_{F \text{ is a type-2 hyperedge}} l_i^F.$$

3. For any *type-3* hyperedge $F = \sigma(t_i)$, since $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F) = 1$, we have that $\Upsilon|_F$ is $(3k^2, k)$-unbreakable and $y[t_i, \Upsilon|_F, \mathbf{v}, \mu^F + \mu', l_1^F + l_1', l_2^F + l_2'] = 1$, where $\mu' = |\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1) \cap F|$, $l_1' = |\{(u, v) \in E(G[\sigma(t_i)]) | \widehat{\Upsilon}_{\mathbf{v}}(u) = \widehat{\Upsilon}_{\mathbf{v}}(v) = V_1\}|$ and $l_2' = |\{(u, v) \in E(G[\sigma(t_i)]) | \widehat{\Upsilon}_{\mathbf{v}}(u) = \widehat{\Upsilon}_{\mathbf{v}}(v) = V_2\}|$.

   We thus derive that there exists a witnessing coloring $\Upsilon^i : \gamma(t_i) \to [k]_0$ for the condition $y[t_i, \Upsilon|_F, \mathbf{v}, \mu^F + \mu', l_1^F + l_1', l_2^F + l_2'] = 1$. Specifically, the following conditions are satisfied.

   (a) Υ^i extends $\Upsilon|_F$.

   (b) $|\widehat{\Upsilon^i}_{\mathbf{v}}^{-1}(V_1)| = \mu^F + \mu'$.

   (c) $A \cap \gamma(t_i) \subseteq \widehat{\Upsilon^i}_{\mathbf{v}}^{-1}(V_1)$ and $B \cap \gamma(t_i) \subseteq \widehat{\Upsilon^i}_{\mathbf{v}}^{-1}(V_2)$.

   (d) $|E(G[\widehat{\Upsilon^i}_{\mathbf{v}}^{-1}(V_1)])| \leq l_1^F + l_1'$, and $|E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)])| \leq l_2^F + l_2'$.

   (e) $\bigcup_{\ell, j \in [k]_0, \ell \neq j} E(\Upsilon^{i-1}(\ell), \Upsilon^{i-1}(j)) = E(G[\widehat{\Upsilon^i}_{\mathbf{v}}^{-1}(V_1)]) \cup E(G[\widehat{\Upsilon^i}_{\mathbf{v}}^{-1}(V_2)])$.

Keeping the above items in mind, we proceed to identify a witnessing coloring for $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$. We construct such a coloring $\Upsilon' : \gamma(t) \to [k]_0$ as follows. For all $v \in \gamma(t)$, if $v \in \beta(t)$, then define $\Upsilon'(v) = \Upsilon(v)$, and otherwise there exists a unique child t_i of t such that $v \in \gamma(t_i)$, in which case we define $\Upsilon'(v) = \Upsilon^i(v)$. For the sake of clarity, let us extract the required argument to the proof of a separate claim.

Claim 12.4.4. *The aforementioned Υ' is a witnessing coloring for $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$.*

Proof. First, note that by Item 1 in the proof of Lemma 12.4.2, we have that $\Upsilon_\sigma \subseteq \Upsilon$ and therefore $\Upsilon_\sigma \subseteq \Upsilon'$. Let us now verify that all the conditions specified in Definition 12.4.5 are satisfied.

- Let us first prove Condition 1. To this end, we observe that by Items 1, 3a and 3b, we have that the three following equalities hold.
  - $|\widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_1)| = |\widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_1) \cap \beta(t)| + \sum_{t_i \text{ is a child of } t \text{ in } T} |\widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_1) \cap (\gamma(t_i) \setminus \sigma(t_i))|$.
  - $|\widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_1) \cap \beta(t)| = |\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1) \cap \beta(t)| = \sum_{F \text{ is a type-1 hyperedge}} \mu^F$.
  - For every child t_i of t, $|\widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_1) \cap (\gamma(t_i) \setminus F)| = \mu^F$, where $F = \sigma(t_i)$.

  Thus, since $\sum_{F \text{ is a type-2 hyperedge}} \mu^F = 0$, we conclude that $|\widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_1)| = \sum_{F \in E(H)} \mu^F = \mu$.
- Next, we prove Condition 2. However, by Items 1 and 3c, we directly deduce that both $A \cap \gamma(t) \subseteq \widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_1)$ and $B \cap \gamma(t) \subseteq \widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_2)$ as required.
- We now turn to prove Condition 3. First observe that there are no edges between a vertex of $\beta(t) \setminus \sigma(t_i)$ and a vertex of $\gamma(t_i) \setminus \sigma(t_i)$. In light of Item 3a, note that

$$\begin{aligned} |E(G[\widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_1)])| &= |E(G[\widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_1) \cap \beta(t)])| + \\ &\quad \sum_{t_i \text{ is a child of } t \text{ in } T} |E(G[\widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_1) \cap \gamma(t_i)])| - \\ &\quad \sum_{t_i \text{ is a child of } t \text{ in } T} |E(G[\widehat{\Upsilon'}_{\mathbf{v}}^{-1}(V_1) \cap \sigma(t_i)])|. \end{aligned}$$

  Now, observe that by Items 2, 3a and 3d, the two following equations hold.

– $|E(G[\widehat{\Upsilon}'^{-1}_{\mathbf{v}}(V_1) \cap \beta(t)])| = |E(G[\widehat{\Upsilon}^{-1}_{\mathbf{v}}(V_1) \cap \beta(t)])| \leq \sum_{F \text{ is a type-2 hyperedge}} l_1^F$.

– For every child t_i of t, $|E(G[\widehat{\Upsilon}'^{-1}_{\mathbf{v}}(V_1) \cap \gamma(t_i)])| = l_1^F + |E(G[\widehat{\Upsilon}'^{-1}_{\mathbf{v}}(V_1) \cap \sigma(t_i)])|$, where $F = \sigma(t_i)$.

Since $\sum_{F \text{ is a type-1 hyperedge}} l_1^F = 0$, we conclude that

$$|E(G[\widehat{\Upsilon}'^{-1}_{\mathbf{v}}(V_1)])| \leq \sum_{F \in E(H)} l_1^F \leq l_1.$$

Similarly, we derive that $|E(G[\widehat{\Upsilon}'^{-1}_{\mathbf{v}}(V_2)])| \leq \sum_{F \in E(H)} l_2^F \leq l_2$.

- Finally, we prove Condition 4. In the first direction, consider some edge $e \in E(G[\widehat{\Upsilon}'^{-1}_{\mathbf{v}}(V_1)]) \cup E(G[\widehat{\Upsilon}'^{-1}_{\mathbf{v}}(V_2)])$. Let us denote $e = \{u, v\}$, and observe that $\widehat{\Upsilon}'_{\mathbf{v}}(v) = \widehat{\Upsilon}'_{\mathbf{v}}(u)$. If $u, v \in \gamma(t_i)$ for some child t_i of t, then by Item 3e, we have that $e \in \bigcup_{\substack{i,j \in [k]_0, \\ i \neq j}} E(\Upsilon'^{-1}(i), \Upsilon'^{-1}(j))$. Otherwise, from point 1 of Theorem 12.4.1, $u, v \in \beta(t)$, and thus e is some type-2 hyperedge F. Since $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F) = 1$, the definition of $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F)$ directly implies that $\Upsilon(u) \neq \Upsilon(v)$, and therefore again $e \in \bigcup_{\substack{i,j \in [k]_0, \\ i \neq j}} E(\Upsilon'^{-1}(i), \Upsilon'^{-1}(j))$.

  In the other direction, consider some edge $e \in \bigcup_{\substack{i,j \in [k]_0, \\ i \neq j}} E(\Upsilon'^{-1}(i), \Upsilon'^{-1}(j))$. Let us denote $e = \{u, v\}$, and observe that $\Upsilon'(v) \neq \Upsilon'(u)$. If $u, v \in \gamma(t_i)$ for some child t_i of t, then by Item 3e, we have that $e \in E(G[\widehat{\Upsilon}'^{-1}_{\mathbf{v}}(V_1)]) \cup E(G[\widehat{\Upsilon}'^{-1}_{\mathbf{v}}(V_2)])$. Otherwise, from point 1 of Theorem 12.4.1, $u, v \in \beta(t)$, and thus e is some type-2 hyperedge F. Since $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F) = 1$, the definition of $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F)$ directly implies that $\widehat{\Upsilon}'_{\mathbf{v}}(v) = \widehat{\Upsilon}'_{\mathbf{v}}(u)$, and therefore again $e \in E(G[\widehat{\Upsilon}'^{-1}_{\mathbf{v}}(V_1)]) \cup E(G[\widehat{\Upsilon}'^{-1}_{\mathbf{v}}(V_2)])$.

Thus, we have proved that Υ' is a witnessing coloring for $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$. Moreover, Υ', which extends Υ, is the desired function for the second part of the lemma. □

This concludes the proof of the lemma. □

In light of Lemma 12.4.2, we now turn to verify that $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F \in E(H)})$ is of the form that we are actually able to solve.

Lemma 12.4.3. $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F\in E(H)})$ *is a favorable instance of* HP.

Proof. First note that $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F\in E(H)})$ is indeed a favorable instance of HP. This is clear from the construction of $\{f_F\}|_{F\in E(H)}$ and the fact that each edge of $F \in E(H)$ has size at most η because of point 3 of Theorem 12.4.1. Let us now verify that each of the three properties of a favorable instance is satisfied.

- **Local Unbreakability:** Let us choose an arbitrary $F \in E(H)$. If F is a type-1 or a type-2 hyperedge, then since $|F| \leq 2$, we have that local unbreakability is trivially satisfied. Otherwise, if F is a type-3 hyperedge, then the satisfaction of local unbreakability directly follows from the construction of f_F.

- **Connectivity:** Choose an arbitrary $F \in E(H)$ along with a tuple (Γ, μ, l_1, l_2) in the domain of f_F such that $f_F(\Gamma, \mu, l_1, l_2) = 1$. If F is a type-1 hyperedge, then connectivity trivially holds. If F is a type-2 hyperedge, then connectivity follows from the construction of f_F. Indeed, to see this, let us denote $F = \{u, v\}$. Then, if $\Gamma(u) \neq \Gamma(v)$, by the second through last case in the definition of f_F, we deduce that $\widehat{\Gamma}_{\mathbf{v}}(u) = \widehat{\Gamma}_{\mathbf{v}}(v)$, else we contradict the supposition that $f_F(\Gamma, \mu, l_1, l_2) = 1$. Then, connectivity directly follows from the third and fourth cases.

  Now, suppose that $F = \sigma(\widehat{t})$ is a type-3 hyperedge, and say $\Gamma : F \to [k]_0$ is such that there exist $i, j \in [k]_0$, $i \neq j$, satisfying $|\Gamma^{-1}(i)| > 0$ and $|\Gamma^{-1}(j)| > 0$. We need to show that $l_1 + l_2 \geq 1$. Since $f_F(\Gamma, \mu, l_1, l_2) = 1$, it holds that $y[\widehat{t}, \Gamma, \mathbf{v}, \mu + \mu', l_1 + l'_1, l_2 + l'_2] = 1$, where μ', l'_1 and l'_2 are as defined at the construction of f_F. Let $\Upsilon : \gamma(\widehat{t}) \to [k]_0$ denote some witnessing coloring for this condition. Since (T, β) is a *highly connected tree decomposition*, the Property 1 of such a decomposition implies that $G^* = G[\gamma(\widehat{t})] \setminus E(G[\sigma(\widehat{t})])$ is connected and that every vertex in $\sigma(\widehat{t})$ is adjacent in G to some vertex in $\gamma(\widehat{t}) \setminus \sigma(\widehat{t})$. Since only the edges internal to $\sigma(\widehat{t})$ were removed in forming G^*, it follows that every two vertices in $\sigma(\widehat{t})$ are connected by a path in G^* Let $u \in \Gamma^{-1}(i)$ and $v \in \Gamma^{-1}(j)$. Note that $u \neq v$ and $i \neq j$. Since u and v are connected by a path in G^*, we derive that G^* has an edge e such that

$$e \in \left(\bigcup_{c,d\in[k]_0, c\neq d} E(\Upsilon^{-1}(c), \Upsilon^{-1}(d)) \right) \setminus E(G[\sigma(t')]).$$

  Recall that $\bigcup_{\substack{c,d\in[k]_0,\\ c\neq d}} E(\Upsilon^{-1}(c), \Upsilon^{-1}(d)) = E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)]) \cup E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)])$.

Therefore, we have that $e \in (E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)]) \cup E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)])) \setminus E(G[\sigma(\widehat{t})])$. Thus, $l_1 + l_2 \geq 1$.

- **Global Unbreakability:** Suppose that $\texttt{aHP}[\mu, l_1, l_2] = 1$. Then, by Lemma 12.4.2, there exists $\Upsilon' : \gamma(t) \to [k]_0$ satisfying the properties listed in that lemma. From here, we get that $\sum_{i,j \in [k]_0, i<j} |E(\Upsilon'^{-1}(i), \Upsilon'^{-1}(j))| \leq l_1 + l_2 \leq k_1 + k_2 \leq k$. We argue that $\Upsilon'|_{\beta(t)}$ is a witnessing coloring for global unbreakability, that is, this function is (q, k)-unbreakable. In this context, we remind the reader that $q = (\eta + k)k$. To prove our argument, we first prove the following claim.

  Claim 12.4.5. *Suppose that there exists $i \in [k]_0$ such that $|\Upsilon'^{-1}(i) \cap \beta(t)| > \eta + k$. Then, $\sum_{j \in [k]_0, i \neq j} |\Upsilon'^{-1}(j) \cap \beta(t)| \leq \eta + k$.*

  Proof. Suppose that the claim is false. Then, both $|\Upsilon'^{-1}(i) \cap \beta(t)| > \eta + k$ and $\sum_{j \in [k]_0, i \neq j} |\Upsilon'^{-1}(j) \cap \beta(t)| > \eta + k$. Thus,

  $$\left(X = \Upsilon'^{-1}(i) \cap \beta(t), Y = \left(\bigcup_{j \in [k]_0, i \neq j} \Upsilon'^{-1}(j) \cap \beta(t) \right) \cup \delta(\Upsilon'^{-1}(i) \cap \beta(t)) \right)$$

  is a separation of order at most k of $G[\gamma(t)]$ as we have already shown that

  $$\sum_{i,j \in [k]_0, i \leq j} |E(\Upsilon'^{-1}(i), \Upsilon'^{-1}(j))| \leq l_1 + l_2 \leq k_1 + k_2 \leq k.$$

  Moreover, $|(X \setminus Y) \cap \beta(t)| > \eta$ and $|(Y \setminus X) \cap \beta(t)| > \eta$, which contradicts point 2 of Theorem 12.4.1, that $\beta(t)$ is (η, k)-unbreakable in $G[\gamma(t)]$. □

  Thus, if there exist $i \in [k]_0$ as defined in Claim 12.4.5, then we are done. That is, we conclude that $\Upsilon'|_{\beta(t)}$ is (q, k)-unbreakable. Otherwise, for all $i \in [k]_0$, it holds that $|\Upsilon'^{-1}(i)| \leq \eta + k$. In particular, for any $i \in [k]_0$, $\sum_{j \in [k]_0, i \neq j} |\Upsilon'^{-1}(j)| \leq (\eta + k)k = q$. Thus, we again conclude that $\Upsilon'|_{\beta(t)}$ is (q, k)-unbreakable. □

  □

Finally, we turn to address the statement complementary to the one of Lemma 12.4.2.

Lemma 12.4.4. *For all $\mu \in [n]$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, if $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$, then $\mathtt{aHP}[\mu, l_1, l_2] = 1$.*

Proof. Fix some $\mu \in [n]$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$ such that $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$. Our objective is to show that $\mathtt{aHP}[\mu, l_1, l_2] = 1$. To this end, let Υ be a witnessing coloring for $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$. We would like to prove that $\Upsilon|_{\beta(t)}$ is a witnessing coloring for $\mathtt{aHP}[\mu, l_1, l_2] = 1$, which would complete the proof of the lemma. To do so, we proceed as follows.

First, for any hyperedge $F \in E(H)$, let us define μ^F, ${l_1}^F$ and ${l_2}^F$ as follows.

- **If F is a type-1 hyperedge:** Set $\mu^F = 1$ if $\widehat{\Upsilon}_{\mathbf{v}}(F) = V_1$, and $\mu^F = 0$ otherwise. Set $l_1^F = 0$ and $l_2^F = 0$.
- **If $F = \{u, v\}$ is a type-2 hyperedge:** Set $\mu^F = 0$. If $\widehat{\Upsilon}_{\mathbf{v}}(u) \neq \widehat{\Upsilon}_{\mathbf{v}}(v)$ and $\Upsilon(u) = \Upsilon(v)$, set ${l_1}^F = {l_2}^F = 0$. Otherwise, if $\widehat{\Upsilon}_{\mathbf{v}}(u) = \widehat{\Upsilon}_{\mathbf{v}}(v) = V_1$, set ${l_1}^F = 1$ and ${l_2}^F = 0$, and if $\widehat{\Upsilon}_{\mathbf{v}}(u) = \widehat{\Upsilon}_{\mathbf{v}}(v) = V_2$, set ${l_1}^F = 0$ and ${l_2}^F = 1$. The other cases cannot arise. Indeed, since Υ is a witnessing coloring for $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$, we have that
  $$\bigcup_{i,j \in [k]_0, i \neq j} E(\Upsilon^{-1}(i), \Upsilon^{-1}(j)) = E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)]) \cup E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)]).$$
- **If F is a type-3 hyperedge:** Denote $F = \sigma(\widehat{t})$, where $\widehat{t}$ is a child of t in T. Set $\mu^F = |\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1) \cap (\gamma(\widehat{t}) \setminus \sigma(\widehat{t}))|$, $l_1^F = |E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1) \cap \gamma(\widehat{t})])| - |E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1) \cap \sigma(\widehat{t})])|$ and $l_2^F = |E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2) \cap \gamma(\widehat{t})])| - |E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2) \cap \sigma(\widehat{t})])|$.

Let us proceed by proving three claims that would together imply that $\Upsilon|_{\beta(t)}$ is a witnessing coloring for $\mathtt{aHP}[\mu, l_1, l_2] = 1$.

Claim 12.4.6. *Let $\widehat{t}$ be a child of t in T, and let $i \in [k]_0$ be such that $|\Upsilon^{-1}(i) \cap \sigma(\widehat{t})| > 3k$. Then, $\sum_{j \in [k]_0, i \neq j} |\Upsilon^{-1}(j) \cap \sigma(\widehat{t})| \leq 3k$.*

Proof. Suppose, by way of contradiction, that the claim is false. That is, we have that both $|\Upsilon^{-1}(i) \cap \sigma(\widehat{t})| > 3k$ and $\sum_{j \in [k]_0, i \neq j} |\Upsilon^{-1}(j) \cap \sigma(\widehat{t})| > 3k$. Consider the separation (X, Y) of $G[\gamma(t)]$, where $X = \Upsilon^{-1}(i)$ and $Y = (\gamma(t) \setminus \Upsilon^{-1}(i)) \cup \delta(\Upsilon^{-1}(i))$. Observe that $X \cap Y = \delta(\Upsilon^{-1}(i))$. Since Υ is a witnessing coloring for $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$, we have that

$$\bigcup_{i,j \in [k]_0, i \neq j} E(\Upsilon^{-1}(i), \Upsilon^{-1}(j)) = E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)]) \cup E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)])$$

and $|E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)])\cup E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)])| \leq l_1+l_2 \leq k_1+k_2 \leq k$. Therefore, $|\delta(\Upsilon^{-1}(i))| \leq k$, and thus the order of the separation (X, Y) is at most k. Moreover, since $|\Upsilon^{-1}(i) \cap \sigma(\widehat{t})| > 3k$, we have that $|(X \setminus Y) \cap \sigma(\widehat{t})| > 3k - k = 2k$, and since $\sum_{j\in[k]_0, i\neq j} |\Upsilon^{-1}(j) \cap \sigma(\widehat{t})| > 3k$, we also have that $|(Y \setminus X) \cap \sigma(\widehat{t})| > 3k$. This implies that $\sigma(\widehat{t})$ is not $(2k, k)$-unbreakable in $G[\gamma(t)]$, which means that $\sigma(\widehat{t})$ is not $(2k, k)$-unbreakable in $G[\gamma(\text{parent}(\widehat{t}))]$. This is a contradiction to the fact that (T, β) is a highly connected tree decomposition—specifically, it should satisfy Property 3 in Theorem 12.4.1. □

Having Claim 12.4.6 at hand, we now verify that each function f_F assigns 1 to the required tuple.

Claim 12.4.7. *For any $F \in E(H)$, $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F) = 1$.*

Proof. First, note that since Υ is a witnessing coloring for $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$, we have that $\Upsilon \subseteq \Upsilon_\sigma$, $A \cap \gamma(t) \subseteq \widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)$ and $B \cap \gamma(t) \subseteq \widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)$. Thus, from the construction of a type-1 hyperedge F and the corresponding function f_F with respect to $HP(k_1, k_2, n, \eta, q, H, \{f_F\}_{F\in E(H)})$, it is clear that $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F) = 1$. Second, suppose F is a type-2 hyperedge. The specifications of f_F, together with our definition of μ^F, l_1^F and l_2^F, directly implies that $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F) = 1$.

Third, suppose that F is a type-3 hyperedge, and denote $F = \sigma(t_i)$ for some t_i that is a child of t in T. Note that $y[t_i, \Upsilon|_F, \mathbf{v}, \mu^F + \mu', l_1^F + l_1', l_2^F + l_2'] = 1$ because $\Upsilon|_{\gamma(t_i)}$ is a witnessing coloring for this equality, where $\mu' = |\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1) \cap \sigma(\widehat{t})|$, $l_1' = |E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1) \cap \sigma(\widehat{t})])|$ and $l_2' = |E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2) \cap \sigma(\widehat{t})])|$. We now need to show that $\Upsilon|_F$ is $(3k^2, k)$-unbreakable, as then we would be able to conclude that $f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F) = 1$. By Claim 12.4.6, if there exists $i \in [k]_0$ such that $|\Upsilon^{-1}(i) \cap \sigma(\widehat{t})| > 3k$, then we deduce that $\Upsilon|_{\sigma(\widehat{t})}$ is $(3k^2, k)$-unbreakable. Otherwise, for all $i \in [k]_0$, $|\Upsilon^{-1}(i) \cap \sigma(\widehat{t})| \leq 3k$. Hence, for any $i \in [k]_0$, $\sum_{j\in[k]_0, i\neq j} |\Upsilon^{-1}(j) \cap \sigma(\widehat{t})| \leq 3k^2$. Thus, we have proved that $\Upsilon_{|F}$ is $(3k^2, k)$-unbreakable. □

Finally, we present our third claim.

Claim 12.4.8. $\mu = \sum_{F\in E(H)} \mu^F$, $\sum_{F\in E(H)} l_1^F \leq l_1$ *and* $\sum_{F\in E(H)} l_2^F \leq l_2$.

By the property of (T, β) being a tree decomposition, for any two children t_i and t_j of t in T, $\gamma(t_i) \cap \gamma(t_j) \subseteq \beta(t)$, and also by the definition, $\sigma(t_i) \subseteq \beta(t)$ for any child t_i of t. Now, note that $\mu = |\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)|$. Thus, to show that $\mu = \sum_{F\in E(H)} \mu^F$,

it is sufficient to show that $|\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)| = \sum_{F \in E(H)} \mu^F$. However, keeping the above argument in mind, the claim that $|\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)| = \sum_{F \in E(H)} \mu^F$ directly follows from the satisfaction of the three following conditions. We remark that the satisfaction of these conditions is a direct consequence of the supposition that Υ is a witnessing coloring for $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$, together with our definition of the values μ^F, l_1^F and l_2^F.

1. For any type-1 hyperedge F, we have that $\mu^F = 1$ only if $\widehat{\Upsilon}_{\mathbf{v}}(F) = V_1$. In particular, $\sum_{F \in E(H) \text{ of type-1}} \mu^F = |\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1) \cap \beta(t)|$.
2. For any type-2 hyperedge F, $\mu^F = 0$. Thus, $\sum_{F \in E(H) \text{ of type-2}} \mu^F = 0$.
3. For any type-3 hyperedge $F = \sigma(t_i)$, $\mu^F = |\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1) \cap (\gamma(t_i) \setminus \sigma(t_i))|$.

Similarly, let us observe that $|E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)])| \leq l_1$. Thus, to show that $\sum_{F \in E(H)} l_1^F \leq l_1$, it is sufficient to show that $\sum_{F \in E(H)} l_1^F \leq |E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)])|$. However, the latter inequality directly follows from the satisfaction of all of the following conditions.

1. For any type-1 hyperedge F, $l_1^F = 0$. Thus, $\sum_{F \in E(H) \text{ of type-1}} l_1^F = 0$.
2. For any type-2 hyperedge $F = \{u, v\}$, ${l_1}^F = 1$ only if $\widehat{\Upsilon}_{\mathbf{v}}(u) = \widehat{\Upsilon}_{\mathbf{v}}(v) = V_1$. In particular, $\sum_{F \in E(H) \text{ of type-2}} l_1^F = |E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)]) \cap E(G[\beta(t)])|$.
3. For any type-3 hyperedge $F = \sigma(t_i)$, $|E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1) \cap (\gamma(t_i) \setminus \sigma(t_i))])| \leq l_1^F$.

Symmetrically, $\sum_{F \in E(H)} l_2^F \leq l_2$. This concludes the proof of the claim. $\diamond$

As we have proved Claims 12.4.7 and 12.4.8, we derive that $\Upsilon|_{\beta(t)}$ is a witnessing coloring for $\texttt{aHP}[\mu, l_1, l_2] = 1$. This concludes the proof of the lemma. $\square$

Recall that we have argued that to prove Theorem 12.3.1, it is sufficient to show that the current set of values $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$ can be computed in time $2^{k^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$. Here, n refers to $|V(G)|$. By Lemmas 12.4.2 and 12.4.4, this set of values can be derived from the solution of $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F \in E(H)})$. Since $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F \in E(H)})$ is a favorable instance of HP (by Lemma 12.4.3), it can be solved in time $2^{\mathcal{O}(\min(k,q) \log(k+q))} \eta^{\mathcal{O}(k^3)} |E(H)|^{\mathcal{O}(1)} = 2^{k^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$, using Theorem 12.4.2.

12.5 Solving Favorable Instances of Hypergraph Painting

Recall the problem statement of Hypergraph Painting (HP) and the definition of a *favorable* instance of HP from Section 12.4. In this section, we prove Theorem 12.4.2. We prove this theorem in two steps. In the first step we prove Lemma 12.5.1. In the second step, we perform a dynamic programming procedure exploiting the structure given in Lemma 12.5.1.

12.5.1 Color Coding The Instance

Again, recall that our goal is to solve the HP problem on a favorable instance. In this section, given a hypergraph, our goal is to somehow partition the vertex set of the hypergraph such that, if the given instance of HP is a Yes instance, then the witnessing coloring for it does not color the parts of this partition in a very "unpredictable" way. This is formally captured in the conditions of Lemma 12.5.1. Before stating the lemma, we first define what we mean by a sets-colorings tuple.

A *sets-colorings* tuple of a hypergraph H, is a tuple consisting of a partition of $V(H)$, $V(H) = C_0 \uplus C_{11} \uplus \ldots \uplus C_{1a} \uplus C_{21} \uplus \ldots \uplus C_{2b}$ ($C_0, C_{11}, \ldots, C_{1a}, C_{21}, \ldots, C_{2b}$ are called the sets of this tuple), and coloring functions $\Phi_i : C_{1i} \to [k]_0$, for all $i \in [a]$, such that for each $F \in E(H)$, either F is contained in some set of this tuple, or intersects at most 2 sets of this tuple, one of which necessarily being C_0 and the other being one of $\{C_{11}, \ldots, C_{1a}\}$. A sets-colorings tuple looks like $(C_0 \uplus C_{11} \uplus \ldots \uplus C_{1a} \uplus C_{21} \uplus \ldots \uplus C_{2b}, \Phi_1, \ldots, \Phi_a)$.

Lemma 12.5.1. *Let $H = (V(H), E(H))$ be a hypergraph and k, d, x, y, q be positive integers. For each $F \in E(H)$, let $|F| \leq d$. Let $\Upsilon : V(H) \to [k]_0$ be a coloring of $V(H)$ satisfying the following conditions.*

1. *The number of hyperedges $F \in E(H)$, such that F is not monochromatic under Υ, is at most x.*
2. *For each $F \in E(H)$, $\Upsilon|_F$ is (y, k)-unbreakable. This condition is called the* local unbreakability *condition of Υ.*
3. *Υ is (q, k)-unbreakable. This condition is called the* global unbreakability *condition of Υ. Let 0 be the globally dominant color of Υ with respect to this global unbreakability.*

Then, given H, k, d, x, y, q, *one can, in time* $\mathcal{O}(2^{\mathcal{O}(\min(x,q)\log(x+q))} \max\{d,y\}^{\mathcal{O}(\max\{xy,xk\})} \cdot |E(H)|^{\mathcal{O}(1)})$, *find a family of size* $\mathcal{O}(2^{\mathcal{O}(\min(x,q)\log(x+q))} \max\{d,y\}^{\mathcal{O}(\max\{xy,xk\})} \cdot \log^{\mathcal{O}(1)}|E(H)|)$, *consisting of sets-colorings tuples of* H, *such that there exists a tuple* $\boldsymbol{t} = (C_0, C_{11} \uplus \ldots \uplus C_{1a} \uplus C_{21} \uplus \ldots \uplus C_{2b}, \Phi_1, \ldots, \Phi_a)$ *in the family where,*

1. $\Upsilon|_{C_0} = 0$,
2. *for each* $i \in [b]$, $\Upsilon|_{C_{2i}}$ *is monochromatic in* Υ,
3. *for each* $i \in [a]$, *either* $\Upsilon|_{C_{1i}} = 0$, *or* $\Upsilon|_{C_{1i}} = \Phi_i$,

A sets-colorings tuple satisfying the properties mentioned in Lemma 12.5.1 is called a *good* sets-colorings tuple for Υ. The rest of the section is devoted to the proof of Lemma 12.5.1.

Outline of the proof of Lemma 12.5.1 We begin by classifying the hyperedges of H based on Υ. The algorithm highlights a set of hyperedges and the colorings of them as given by Υ using color coding. In the next phase, based on this highlighting, an auxiliary graph is constructed and later tweaked to clean the unwanted highlighting - the side effect of color coding. Eventually another auxiliary graph is constructed which is finally exploited to give the desired output.

12.5.1.1 Classifying Hyperedges

By the global unbreakability of $\Upsilon : V(H) \to [k]_0$, $\sum_{j \in [k]_0, j \neq i} |\Upsilon^{-1}(j)| \leq q$ for some index $i \in [k]_0$. Without loss of generality, suppose that $i = 0$ is such an index, that is, $\sum_{j \in [k]} |\Upsilon^{-1}(j)| \leq q$. We first categorize the hyperedges of H into the following types, based on the coloring Υ. In this context, we remind that the notation $f(A') = b$ indicates that for all $a \in A'$, it holds that $f(a) = b$ (see Section 12.1).

- Let $E_b = \{F \in E(H) : \Upsilon(F) = 0\}$. Here, 'b' stands for *big*.
- For each $i \in [k]$, let $E_{s_i} = \{F \in E(H) : \Upsilon(F) = i\}$. Here, 's' stands for *small*.
- Let $E_m = \{F \in E(H) :$ there exist $u, v \in F$ such that $\Upsilon(u) \neq \Upsilon(v)\}$. Here, 'm' stands for *multichromatic*.

Observe that each hyperedge $F \in E(H)$ belongs to exactly one of the sets $E_b, E_m, E_{s_1}, \ldots, E_{s_k}$. Furthermore, let E'_{s_i} denote the edge set of some arbitrary spanning forest of the hypergraph on the vertex set $V(H)$ and the edge set E_{s_i}. Let $E_s = \bigcup_{i\in[k]} E'_{s_i}$ denote the union of these edge sets. From the properties of Υ, $|E_m| \leq x$. Also, as we will see in Lemma 12.5.2, $|E_s| \leq q$. We exploit these bounds to highlight the hyperedges in E_m and E_s (Lemma 12.5.4) efficiently. In addition to this, as we shall see in Lemma 12.5.3, the total number of possible restrictions of Υ on any hyperedge can also be bounded effectively. Thus, we cannot only highlight the hyperedges in E_m and E_s, but we can also guess the restrictions of Υ to these hyperedges. The proof of Lemma 12.5.4 would capture the idea of the performance of highlighting and guessing. As one would expect, this highlighting does not conclude our arguments, as it does not just highlight the hyperedges in E_m and E_s, but also some hyperedges from E_b. We deal with the inherent challenges of handling such a "messy picture" later in our proof.

Lemma 12.5.2. $|E_s| \leq q$.

Proof. Recall that for each $i \in [k]$, we defined E'_{s_i} as the edge set of a spanning forest of the hypergraph with the vertex set $V(H)$ and the edge set E_{s_i}. Hence, by this definition, $|E'_{s_i}| \leq |\Upsilon^{-1}(i)|$. Now, recall that since Υ is (q,k)- unbreakable, we assumed w.l.o.g. that $\sum_{i\in[k]} |\Upsilon^{-1}(i)| \leq q$. We thus have that $\sum_{i\in[k]} |E'_{s_i}| \leq q$. Therefore, $|E_s| \leq q$. □

12.5.1.2 Introducing Good Assignments

Let us first note that by Lemma 12.1.1, for any hyperedge $F \in E(H)$, the number of (y,k)-unbreakable functions (that we call (y,k)-unbreakable colorings) from F (recall $|F| \leq d$) to $[k]_0$ is at most $\alpha = \sum_{l=1}^{y} \binom{d}{l} \cdot y^k \cdot (k+1) = \max\{d,y\}^{\mathcal{O}(\max\{y,k\})}$. For each hyperedge F, let us arbitrarily order all possible (y,k)-unbreakable colorings. For each $i \in [\alpha]$, let $\lambda_{F,i}$ denote the i-th such coloring. If for an hyperedge F, the number of such colorings is strictly smaller than α, then we extend its list of possible colorings to be of size α by letting some colorings be present multiple times. Thus, for each $F \in E(H)$ and $i \in [\alpha]$, we ensure $\lambda_{F,i}$ is well-defined.

Lemma 12.5.3. *For any $F \in E(H)$, there exists $i \in [\alpha]$ such that $\Upsilon|_F = \lambda_{F,i}$.*

Proof. This follows from the fact that $\Upsilon|_F$ is (y,k)-unbreakable.

□

Here, we are interested in assignments that are functions associating each hyperedge $F \in E(H)$ with a coloring $\lambda_{F,i}$. Let us proceed by defining which assignments would be useful for us to have at hand.

Definition 12.5.1. *An assignment $p : E(H) \to [\alpha]_0$ is said to be a* good *assignment if the following conditions hold.*

1. *For all $F \in E_s$, $p(F) = 0$.*
2. *For all $F \in E_m$, $p(F) = i > 0$ and $\Upsilon|_F = \lambda_{F,i}$.*

To employ color coding, we first mention the required derandomization tools.

Proposition 12.5.1 (Lemma 1.1, [?])**.** *Given a set U of size n and $c, d \in [n]_0$, we can construct in time $\mathcal{O}(2^{\mathcal{O}(\min(c,d)\log(c+d))} n \log n)$ a family $\mathcal{F}$ of at most $\mathcal{O}(2^{\mathcal{O}(\min(c,d)\log(c+d))} \log n)$ subsets of U, such that the following holds: for all sets $C, D \subseteq U$ such that $C \cap D = \emptyset$, $|C| \leq c$ and $|D| \leq d$, there exists a set $S \in \mathcal{F}$ with $C \subseteq S$ and $D \cap S = \emptyset$.*

Definition 12.5.2 ((N,r)-perfect family)**.** *For any universe N, an (N,r)-perfect family is a family of functions from N to $[r]$, such that for any subset $X \subseteq N$ of size r, there exists a function in the family that is injective on X.*

Proposition 12.5.2 ([168])**.** *An (N,r)-perfect family of size $\mathcal{O}(e^r r^{\mathcal{O}(\log r)} \log |N|)$ can be computed in time $\mathcal{O}(e^r r^{\mathcal{O}(\log r)} |N| \log |N|)$.*

We are now ready to present our color coding phases.

Lemma 12.5.4. *There exists a set $\mathcal{A}$ of assignments from $E(H)$ to $[\alpha]_0$, such that $|\mathcal{A}| \leq 2^{\mathcal{O}(\min(x,q)\log(x+q))} \cdot \max\{d,y\}^{\mathcal{O}(\max\{xy,xk\})} \cdot \log^2 |E(H)|$ and there exists a good assignment in $\mathcal{A}$. Moreover, such a set $\mathcal{A}$ is computable in time $\mathcal{O}(2^{\mathcal{O}(\min(x,q)\log(x+q))}$ $\max\{d,y\}^{\mathcal{O}(\max\{xy,xk\})} \cdot |E(H)|^{\mathcal{O}(1)})$.*

Proof. We start by defining three families, which would guide us through the construction of $\mathcal{A}$. For $U = E(H)$, $c = x$ and $d = q$, let $\mathcal{F} = \{S_1, \ldots, S_\nu\}$ be the family of size $\nu = 2^{\mathcal{O}(\min(x,q)\log(x+q))} \log |E(H)|$ obtained by calling the algorithm of Proposition 12.5.1. For each $j \in [\nu]$, let $\mathcal{P}_j$ be a $(E(H) \setminus S_j, x)$-perfect family of size at most $\zeta \leq e^x x^{\mathcal{O}(\log x)} \log |E(H)|$ computed by the algorithm of Proposition 12.5.2. Let $\mathcal{Q}$ be the family of all possible functions from $[x]$ to $[\alpha]$. Observe that $|\mathcal{Q}| = \alpha^x$.

For each set $S_j \in \mathcal{F}$, function $\kappa \in \mathcal{P}_j$ and function $\kappa_0 \in \mathcal{Q}$, let $p[S_j, \kappa, \kappa_0] : E(H) \to [\alpha]_0$ be defined as follows.

$$p[S_j, \kappa, \kappa_0](F) = \begin{cases} 0, & \text{if } F \in S_j \\ \kappa_0(\kappa(F)) & \text{otherwise} \end{cases}$$

Let $\mathcal{A} = \{p[S_j, \kappa, \kappa_0] : S_j \in \mathcal{F}, \kappa \in \mathcal{P}_j, \kappa_0 \in \mathcal{Q}\}$. We claim that there exists a good assignment in $\mathcal{A}$. Since $|E_m| \leq x$ (from the preconditions of Lemma 12.5.1) and $|E_s| \leq q$ (from Lemma 12.5.2), from Proposition 12.5.1 there exists $S_j \in \mathcal{F}$ such that $E_s \subseteq S_j$ and $E_m \cap S_j = \emptyset$. By Proposition 12.5.2, there exists a function $\kappa \in \mathcal{P}_j$ which is injective on E_m. Let $E_m = \{F_1, \ldots, F_c\}$ where $c \leq x$. Without loss of generality, $\kappa(F_y) = y$ for all $y \in [c]$. Since $\mathcal{Q}$ contains all possible functions from $[x]$ to $[\alpha]$, and for each $F \in E_m$ there exists $i \in [\alpha]$ such that $\Upsilon|_F = \lambda_{F,i}$ (from Lemma 12.5.3), there exists $\kappa_0 \in \mathcal{Q}$ such that for each $F \in E_m$, $\Upsilon|_F = \lambda_{F,\kappa_0(\kappa(F))}$. Moreover, since $E_s \subseteq S_j$, we have that $p[S_j, \kappa, \kappa_0](E_s) = 0$. Thus, $p[S_j, \kappa, \kappa_0] \in \mathcal{A}$ is a good assignment.

Recall that $\alpha = \max\{d, y\}^{\mathcal{O}(\max\{y,k\})}$. Now, as we have upper bounded ν and ζ, we observe that $|\mathcal{A}| \leq \nu\zeta\alpha^x = 2^{\mathcal{O}(\min(x,q)\log(x+q))}e^x x^{\mathcal{O}(\log x)}\max\{d, y\}^{\mathcal{O}(\max\{xy,xk\})} \log^2|E(H)|$. This proves the desired bound on the size of $\mathcal{A}$.

The time taken to compute $\mathcal{A}$ is proportional to the time taken to compute $\mathcal{F}, \mathcal{P}_j$ for each $j \in \{\nu\}$ and $\mathcal{Q}$. By Propositions 12.5.1 and 12.5.2, we thus derive that the running time is upper bounded by $\mathcal{O}(2^{\mathcal{O}(\min(x,q)\log(x+q))} \cdot \max\{d, y\}^{\mathcal{O}(\max\{xy,xk\})} \cdot |E(H)|^{\mathcal{O}(1)})$. □

In the next section, we work with a fixed assignment $p \in \mathcal{A}$. For each such assignment, we eventually compute a sets-colorings tuple of H. The family as described in Lemma 12.5.1, is then the union of these tuples for each $p \in \mathcal{A}$. We also prove that if p is a good assignment, then the sets-colorings tuple corresponding to it is a good sets-colorings tuple for Υ. Since, from Lemma 12.5.4, there exists a $p \in \mathcal{A}$, such that p is good, the family of sets-colorings tuples obtained in the end contains a good sets-colorings tuple for Υ.

12.5.1.3 Associating the Graph L_p with an Assignment p

For our assignment $p : E(H) \to [\alpha]_0$, let us now construct an undirected simple graph L_p with $V(L_p) = V(H)$. For each $F \in E(H)$ such that $p(F) = 0$, make F a

clique in L_p. We say that the edges of this clique are the edges that *correspond* to the hyperedge F. For any $F \in E(H)$ such that $p(F) = i > 0$, for each $j \in [k]_0$, make the set ${\lambda_{F,i}}^{-1}(j)$ a clique in L_p. We say that the edges of all such cliques are the edges that *correspond* to the hyperedge F. Since we want L_p to be a simple graph, between any two vertices of L_p we retain at most one copy of the edge between them (if one exists). If a deleted copy of some edge e in L_p corresponds to some hyperedge F, then in the simple graph the retained copy of that edge e is the one that is said to correspond to that hyperedge F (even if we originally added the retained copy of e due to a different hyperedge). Note that it may thus be the case that one edge in L_p corresponds to several hyperedges in $E(H)$.

We proceed by analyzing the connected components of L_p. Informally, we first argue if p is a good assignment, then every connected component of L_p behaves as a single unit with respect to Υ.

Lemma 12.5.5. *Let p be a good assignment and let D be any connected component of L_p. Then, $\Upsilon(D) = i$ for some $i \in [k]_0$, that is, all the vertices in D are assigned the same color by Υ.*

Proof. For any $\mathcal{F} \subseteq E(H)$, let $L_p[\mathcal{F}]$ be the simple graph on the same vertex set as L_p, whose edge set contains only those edges of L_p that correspond to some hyperedge in $\mathcal{F}$. Observe that $L_p[E(H)] = L_p$. Moreover, observe that if a set of vertices is connected in $L_p[\mathcal{F}]$ then it is also connected in $L_p[\mathcal{F}']$ for any $\mathcal{F}' \supseteq \mathcal{F}$.

Let $E(H) = \{F_1, \ldots, F_r\}$. Moreover, for any $j \in [r]$, denote $\mathcal{F}_j = \bigcup_{c=1}^{j} F_c$. Let us prove by induction on j that for each component D of $L_p[\mathcal{F}_j]$, we have that $\Upsilon(D) = i$ for some $i \in [k]_0$. The proof of this claim would conclude the proof of the lemma, as by setting $j = r$, we thus derive that for each component D of $L_p[\mathcal{F}_r] = L_p$, we have that $\Upsilon(D) = i$ for some $i \in [k]_0$. Hence, we next focus only on the proof of the claim.

To prove the base case, where $j = 1$, consider the graph $L_p[\mathcal{F}_1]$. If $F_1 \notin E_m$, then $\Upsilon(F_1) = i$ for some $i \in [k]_0$ (by the definition of E_m). Hence, for each connected component D of $L_p[\mathcal{F}_1]$, $\Upsilon(D) = i$ for some $i \in [k]_0$. Otherwise, $F_1 \in E_m$. In this case, let $p(F_1) = s > 0$. Since p is a good assignment, $\lambda_{F_1,s} = \Upsilon|_{F_1}$. Since each component D of $L_p[\mathcal{F}_1]$ is either an isolated vertex or $\lambda^{-1}_{F_1,s}(i)$ for some $i \in [k]_0$, we conclude that $\Upsilon(D) = i$ for some $i \in [k]_0$.

We now suppose that $j \geq 2$. By induction hypothesis, for each connected component D of $L_p[\mathcal{F}_{j-1}]$, we have that $\Upsilon(D) = i$ for some $i \in [k]_0$. Let us now examine

the graph $L_p[\mathcal{F}_j]$ and the hyperedge F_j. Note that $F_j = \mathcal{F}_j \setminus \mathcal{F}_{j-1}$. If $F_j \notin E_m$, then $\Upsilon(F_j) = i$ for some $i \in [k]_0$ (from the definition of E_m). Let $\mathcal{D}$ be the collection of every connected components of $L_p[\mathcal{F}_{j-1}]$ which intersects F_j. Then, the definition of L_p and the inductive hypothesis directly imply that $\Upsilon(\bigcup \mathcal{D}) = i$ for some $i \in [k]_0$. Thus, by the inductive hypothesis, for each connected component D of $L_p[\mathcal{F}_j]$, we have that $\Upsilon(D) = i$ for some $i \in [k]_0$. Otherwise, $F_j \in E_m$. Then, denote $p(F_1) = s > 0$. Since p is a good assignment, $\lambda_{F_1,s} = \Upsilon|_{F_1}$. For each $i \in [k]_0$, let $\mathcal{D}_i$ be the collection of all connected components of $L_p[\mathcal{F}_{j-1}]$ that intersect $\lambda^{-1}_{F_j,s}(i)$. Then, the definition of L_p and the inductive hypothesis directly imply $\Upsilon(\mathcal{D}_i) = i$. Hence, by the inductive hypothesis, for each connected component D of $L_p[\mathcal{F}_j]$, we have that $\Upsilon(D) = i$ for some $i \in [k]_0$. □

Roughly speaking, we now show that given a good assignment p, if a hyperedge F of H intersects multiple components of L_p and, Υ assigns a color $i > 0$ to at least one of the components, then $F \in E_m$.

Lemma 12.5.6. *Let p be a good assignment and let D be any connected component of L_p such that $\Upsilon(D) = i > 0$ for some $i \in [k]$. For any $F \in E(H)$ such that $F \cap D \neq \emptyset$ and $F \setminus D \neq \emptyset$, then $F \in E_m$.*

Proof. Suppose that the statement is false, that is, there exists $F \in E(H) \setminus E_m$ such that $F \cap D \neq \emptyset$ and $F \setminus D \neq \emptyset$. Since $F \notin E_m$, $F \cap D \neq \emptyset$ and $\Upsilon(D) > 0$, there exists $j \in [k]$ such that $F \in E_{s_j}$. Since $F \cap D \neq \emptyset$ and $\Upsilon(D) = i$, we have that $j = i$, that is, $F \in E_{s_i}$. Recall that E'_{s_i} is a spanning forest of the hypergraph with vertex set $V(H)$ and edge set E_{s_i}. Observe that, since p is a good assignment, by the definition of L_p, for any spanning forest E'_{s_i}, all vertices of F lie in the same component of L_p, which contradicts that $F \setminus D \neq \emptyset$. □

12.5.1.4 Rules to Modify a Good Assignment

We now modify the assignment p by applying the following rule exhaustively. Note that whenever we change p, we update L_p accordingly.

> **Rule 1**: If there exist a connected component D of L_p and a hyperedge $F \in E(H)$ such that $F \subseteq D$ and $p(F) > 0$, then update $p(F) = 0$.

Lemma 12.5.7. *If p was a good assignment, then after any application of Rule 1, it remains a good assignment.*

Proof. From Lemma 12.5.5, $\Upsilon(D) = i$ for some $i \in [k]_0$. Thus, if $F \subseteq D$, then $F \notin E_m$. Hence, when we redefine $p(F) = 0$, p remains a good assignment. □

For each connected component D of L_p, let us now define a *label set* $L(D) \subseteq [k]_0$ as follows. For any $i \in [k]_0$, we insert i into $L(D)$ if and only if there exists $F \in E(H)$ such that $F \cap D \neq \emptyset$, $p(F) = j > 0$ and $\lambda_{F,j}(F \cap D) = i$. Observe that $L(D)$ could be empty.

Let us now turn to analyze the label sets we have just defined.

Lemma 12.5.8. *For any assignment p, let D be a connected component of L_p such that $L(D) = \emptyset$. Then, for any $F \in E(H)$ such that $F \cap D \neq \emptyset$, $F \setminus D = \emptyset$.*

Proof. Observe that if there exists $F \in E(H)$ such that $p(F) > 0$ and $F \cap D \neq \emptyset$, then $|L(D)| \geq 1$. Therefore, if $L(D) = \emptyset$, then for all $F \in E(H)$ such that $F \cap D \neq \emptyset$, we have that $p(F) = 0$. Thus, from the construction of L_p, we have that $F \setminus D = \emptyset$. □

Lemma 12.5.9. *Let p be a good assignment such that Rule 1 is no longer applicable to L_p. Then, for any connected component D of L_p, if $\Upsilon(D) = i > 0$, then either $L(D) = \emptyset$ or $L(D) = \{i\}$.*

Proof. Suppose that $L(D) \neq \emptyset$. Then, there exists $F \in E(H)$ such that $F \cap D \neq \emptyset$ and $p(F) = j > 0$. Let $\lambda_{F,j}(F \cap D) = s$. We will now show that $s = i$. First of all, let us argue that $F \setminus D \neq \emptyset$. Indeed, if $F \setminus D = \emptyset$, then $F \subseteq D$. In this case, since p is a good assignment, where Rule 1 has been exhaustively applied, $p(F)$ should be equal to 0, which is a contradiction. Thus, since $\Upsilon(D) = i > 0$, $F \cap D \neq \emptyset$ and $F \setminus D \neq \emptyset$, from Lemma 12.5.6, we have that $F \in E_m$. Then, since p is a good assignment, $\lambda_{F,j}(F \cap D) = \Upsilon|_{F \cap D}$. Since $\Upsilon(D) = i$, we derive that indeed $\lambda_{F,j}(F \cap D) = i$. Thus, $L(D) = \{i\}$. □

By Lemma 12.5.9, we have that if p is a good assignment and D is a connected component of L_p such that either $L(D) = \{0\}$ or $|L(D)| \geq 2$, then $\Upsilon(D) = 0$.

Lemma 12.5.10. *If p is a good assignment such that Rule 1 is no longer applicable to L_p, and D is a connected component of L_p such that $L(D) = \{l_d\}$, then either $\Upsilon(D) = l_d$ or $\Upsilon(D) = 0$.*

Proof. Since $L(D) = \{l_d\}$, there exists $F \in E(H)$ such that $p(F) = i > 0$, $F \cap D \neq \emptyset$ and $\lambda_{F,i}(F \cap D) = l_d$. Since Rule 1 has been applied exhaustively, $F \setminus D \neq \emptyset$. Denote

$\Upsilon(D) = j$, and suppose that $j \neq 0$, else we are done. Since $j \neq 0$, from Lemma 12.5.6 we have that $F \in E_m$. Then, since p is a good assignment, $\lambda_{F,i} = \Upsilon|_F$. Finally, since all the vertices of D are assigned the same color by Υ (by Lemma 12.5.5), we have that $\Upsilon(D) = l_d$. □

For a connected component D of L_p such that $|L(D)| \geq 2$, let us redefine the label set of D to be $L(D) = \{0\}$. Now, for any connected component D of L_p, $|L(D)| \leq 1$. Moreover, if p is a good assignment and $L(D) = \{0\}$, then $\Upsilon(D) = 0$ (by Lemma 12.5.9). We call a connected component D of L_p such that $L(D) = \{0\}$ a 0-*component*. Thus, from Lemma 12.5.9, if p is a good assignment and D is a 0-component of L_p, then $\Upsilon(D) = 0$.

Let us continue modifying the assignment p, now with the following rule. Again, whenever we modify p, we update L_p accordingly.

Rule 2: If there exist $F \in E(H)$ and two distinct 0-components of L_p, D_1 and D_2, such that $F \cap D_1 \neq \emptyset$ and $F \cap D_2 \neq \emptyset$, then update $p(F) = 0$.

Lemma 12.5.11. *If p is a good assignment then after the application of Rule 2, it remains good.*

Proof. To prove the lemma, it is sufficient to show that $F \notin E_m$. Suppose that this claim is false, that is, $F \in E_m$ and hence after the update, we obtain an assignment that is not good. Since (the original) p is a good assignment, we have that $p(F) = i > 0$ such that $\lambda_{F,i} = \Upsilon|_F$. Since D_1 and D_2 are different connected components of L_p, $(F \cap D_1) \subseteq \lambda_{F,i}^{-1}(j_1)$, $(F \cap D_2) \subseteq \lambda_{F,i}^{-1}(j_2)$ and $j_1 \neq j_2$. However, since D_1 and D_2 are 0-components of L_p, $\Upsilon(D_1) = 0$ and $\Upsilon(D_2) = 0$. Hence, $\lambda_{F,i}(F \cap (D_1 \cup D_2)) = 0$, and so, $F \cap (D_1 \cup D_2)$ is a clique in L_p. This contradicts that D_1 and D_2 are two different components of L_p. □

To further analyze 0-components, define B as the set containing every vertex $v \in V(H)$ such that $\Upsilon(v) = 0$ and there exists $F \in E_m$ that is incident to v.

Lemma 12.5.12. *Let p be a good assignment and let D be a connected component of L_p containing a vertex $v \in B$. Then, D is a 0-component.*

Proof. From the definition of the set B, there exists $F \in E_m$ such that $v \in F$. Since p is a good assignment, $p(F) = i > 0$ such that $\lambda_{F,i} = \Upsilon|_F$. Since $\Upsilon(v) = 0$, $v \in F$

and $v \in D$, we have that $\lambda_{F,i}(F \cap D) = 0$. Hence, $0 \in L(D)$. Therefore, by Lemma 12.5.9, we conclude that D is a 0-component of L_p. □

12.5.1.5 Constructing a Supergraph L_p^* of L_p

Let us now construct another simple undirected graph L_p^*, which is a supergraph of L_p with the same vertex set as of L_p and the following additional edges. If there exists $F \in E(H)$ and two distinct connected components of L_p, D_1 and D_2, such that $F \cap D_1 \neq \emptyset$, $F \cap D_2 \neq \emptyset$, $L(D_1) \neq \{0\}$ and $L(D_2) \neq \{0\}$, then insert an edge between some vertex of D_1 and some vertex of D_2 into L_p^*. Clearly, any connected component D of L_p is contained in some connected component of L_p^*. This leads us to the following definition.

Definition 12.5.3. *Given a connected component D^* of L_p^*, we say that a connected component D of L_p is a* constituent *of D^* if $D \subseteq D^*$.*

A component D^* of L_p^* is called a 0-component of L_p^* if it has only one constituent component and that constituent component is a 0-component in L_p. We now proceed to analyze the new graph L_p^*.

Lemma 12.5.13. *Let D^* be some connected component of L_p^* that has a constituent component D such that $L(D) = \emptyset$ or $L(D) = \{0\}$. Then, D is the only constituent component of D^*, that is, $D^* = D$.*

Proof. When $L(D) = \{0\}$, the lemma follows from the construction of L_p^*. When $L(D) = \emptyset$, by Lemma 12.5.8, for any $F \in E(H)$ such that $F \cap D \neq \emptyset$, we have that $F \setminus D = \emptyset$. Thus, by the construction of L_p^*, it holds that $D^* = D$. □

From Lemma 12.5.13, we have that a component of L_p^* is a 0-component of L_p^* if and only if it is a 0-component of L_p.

Lemma 12.5.14. *Let D^* be a connected component of L_p^*. Let D be some constituent component of D^*. If $\Upsilon(D) = 0$, then $\Upsilon(D^*) = 0$.*

Proof. If $D^* = D$, then we are done. Otherwise, for the sake of contradiction, suppose that $\Upsilon(D^*) \neq 0$. Then there exists a constituent component $\tilde{D}$ of D^* such that $\Upsilon(\tilde{D}) \neq 0$. Since $\Upsilon(D) = 0$ and D^* is connected, there exists constituent components D', D'' of D^*, $D' \neq D''$, such that there is an edge between D' and D'' in D^* and, $\Upsilon(D') = 0$ and $\Upsilon(D'') \neq 0$. Since there is an edge between D' and D'' in

D^*, from the construction of L_p^*, there exists $F \in E(H)$ such that $F \cap D' \neq \emptyset$ and $F \cap D'' \neq \emptyset$. Since $\Upsilon(D') = 0$, $\Upsilon(D'') \neq 0$, $F \cap D' \neq \emptyset$ and $F \cap D'' \neq \emptyset$, $F \in E_m$. Since p is a good assignment, from the construction of L_p and the assigning of label sets, $L(D') = \{0\}$. From Lemma 12.5.13, this implies that D' is the only constituent component of D^*, which is a contradiction. □

Lemma 12.5.15. *For any $F \in E(H)$, F intersects exactly one non 0-component of L_p^*.*

Proof. If there exists $F \in E(H)$ which intersects two non 0-components of L_p^*, then from the construction of L_p^*, those two components are joined by an edge in L_p^* and hence, are the same component in L_p^*. If there exists $F \in E(H)$ which intersects two 0-components of L_p^*, then this violates that Rule 2 has been applied. □

We are now ready to output the sets-colorings tuple $\mathbf{t}_p = (C_0 \uplus C_{11} \uplus \ldots \uplus C_{1a} \uplus C_{21} \uplus \ldots \uplus C_{2b}, \Phi_1, \ldots, \Phi_a)$ corresponding to the assignment p. The sets of $\mathbf{t}_p$ correspond to the connected components of L_p^*, C_0 is the collection of the 0-components of L_p^*, $\{C_{11}, \ldots, C_{1a}\}$ are the components of L_p^* whose constituents have non-empty label set and $\{C_{21}, \ldots, C_{2b}\}$ are the components of L_p^* whose unique constituent has an empty label set. For any $i \in [a]$, $\Phi_i : C_{1i} \to [k]_0$ is defined as follows. Since C_{1i} is a connected component of L_p^*, let $C_{1i}^1, C_{1i}^2, \ldots, C_{1i}^j$ be its constituent components. Then for any $r \in [j]$, $\Phi_i(C_{1i}^r) = L(C_{1i}^r)$ (recall $L(C_{1i}^r)$ has a unique label for the constituent component C_{1i}^r). From Lemma 12.5.15, each hyperedge intersects at most one of $\{C_{11}, \ldots C_{1a}, C_{21}, \ldots, C_{2b}\}$. Also, from Lemma 12.5.9 and Lemma 12.5.8, if $F \cap C_{2i} \neq \emptyset$, then $F \setminus C_{2i} = \emptyset$. This, proves that the tuple $\mathbf{t}_p$ is indeed a sets-colorings tuple for H.

We will now prove that if p is a good assignment, then $\mathbf{t}_p$ is the tuple with the properties desired in Lemma 12.5.1.

Lemma 12.5.16. *If p is a good assignment, then the sets-colorings tuple $\mathbf{t}_p$ corresponding to it is a good tuple for Υ.*

Proof. We show that $\mathbf{t}_p$ satisfies all the properties described in Lemma 12.5.1.

1. By the definition of C_0, 0-component of L_p^*, 0-component of L_p and Lemma 12.5.9, $\Upsilon|_{C_0} = 0$.
2. From Lemma 12.5.13 and the definition of C_{2i}, C_{2i} is some connected component of L_p. Thus, from Lemma 12.5.5, $\Upsilon|_{C_{2i}}$ is monochromatic in Υ.

3. Consider any C_{1i}. From the construction of C_{1i}, C_{1i} is a connected component of L_p^* each of whose constituent components have a non-empty label set. Thus, from Lemma 12.5.10 and 12.5.14, either $\Upsilon|_{C_{1i}} = 0$, or $\Upsilon|_{C_{1i}} = \Phi_i$.

□

Thus, the algorithm for Lemma 12.5.1, for each $p \in \mathcal{A}$, constructs L_p^* and computes a corresponding sets-colorings tuple as discussed before. Here, $\mathcal{A}$ is the family in Lemma 12.5.4. It then outputs the family containing these sets-colorings tuples for each $p \in \mathcal{A}$. From Lemma 12.5.4, there exists a $p \in \mathcal{A}$, such that p is a good assignment. Also, from Lemma 12.5.16, if p is a good assignment then $\mathbf{t}_p$ is a good tuple for Υ. Thus, the output family of sets-colorings tuples contains a good tuple for $\mathbf{t}_p$. We are now left to analyse the running time of the algorithm.

Running Time Analysis. The algorithm begins by computing a family $\mathcal{A}$ of assignments from $E(H)$ to $[\alpha]_0$ using Lemma 12.5.4. Then for each assignment $p \in \mathcal{A}$, the algorithm constructs the graph L_p (in polynomial time), modifies it using Rule 1 and 2 (in polynomial time), assigns it labels (in polynomial time), constructs L_p^* (in polynomial time) and finally constructs the sets-colorings tuple for it (in polynomial time). Since, from Lemma 12.5.4, $|\mathcal{A}| = \mathcal{O}(2^{\mathcal{O}(\min(x,q)\log(x+q))}\max\{d,y\}^{\mathcal{O}(\max\{xy,xk\})} \cdot \log^2|E(H)|)$ and the time taken to compute $|\mathcal{A}|$ is $\mathcal{O}(2^{\mathcal{O}(\min(x,q)\log(x+q))} \max\{d,y\}^{\mathcal{O}(\max\{xy,xk\})} \cdot |E(H)|^{\mathcal{O}(1)})$, the total time taken by the algorithm is $\mathcal{O}(2^{\mathcal{O}(\min(x,q)\log(x+q))} \cdot \max\{d,y\}^{\mathcal{O}(\max\{xy,xk\})} \cdot |E(H)|^{\mathcal{O}(1)})$.

12.5.2 Dynamic Programming

Recall that our aim is to prove Theorem 12.4.2, that is, we need to design an algorithm to solve favorable instances of HP. To do so, we will use Lemma 12.5.1 followed by a dynamic programming procedure for each sets-colorings tuple in the family returned by the algorithm of Lemma 12.5.1. Recall a favorable instance of HP, $I = (k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$. If $\texttt{aHP}[\mu, l_1, l_2] = 1$, then there exists a witnessing coloring $\Upsilon : V(H) \to [k]_0$ for $\texttt{aHP}[\mu, l_1, l_2]$. We will show that since I is a favorable instance of HP, Υ satisfies the prerequisites of Lemma 12.5.1. Then, for each sets-colorings tuple in the family returned by Lemma 12.5.1, we define $k+1$ coloring functions for each hyperedge of H. These coloring functions are defined in such a way, that when we later compute $\texttt{aHP}[\mu, l_1, l_2]$ using dynamic programming, these coloring functions together give a coloring for $V(H)$. Moreover, if $\texttt{aHP}[\mu, l_1, l_2] = 1$,

then since there exists a witnessing coloring Υ for $\texttt{aHP}[\mu, l_1, l_2]$ that satisfies the pre-conditions of Lemma 12.5.1, the dynamic programming procedure corresponding to the sets-colorings tuple that satisfies the conditions of Lemma 12.5.1, will return 1 (or Yes).

Proof of Theorem 12.4.2. Given a favorable instance $I = (k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$ of HP, our algorithm proceeds by calling the algorithm of Lemma 12.5.1 on the instance (H, k, d, x, y, q), where $k = k_1 + k_2, x = k$ and $y = 3k^2$. The output is a family, say, $\mathcal{T}$, of sets-colorings tuples of H.

For each sets-colorings tuple $\mathbf{t} \in \mathcal{T}$, for each $F \in E(H)$, we define $k+1$ coloring functions from F to $[k]_0$, $\Psi_F^1, \ldots, \Psi_F^{k+1}$ (defined later). Let $\mathbf{t} = (C_0 \uplus C_{11} \uplus \ldots \uplus C_{1a} \uplus C_{21} \uplus \ldots \uplus C_{2b}, \Phi_1, \ldots \Phi_a)$. Rename the sets in the tuple $\mathbf{t}$ as $\{S_0, S_1, \ldots, S_z\}$, where $z = a + b$, such that $S_0 = C_0$, for all $i \in [a]$, $S_i = C_{1i}$ and for all $i \in [b]$, $S_{a+i} = C_{2i}$. For each $i \in [z]$ and $j \in [k]_0$, define a function $\Psi_i^j : S_i \to [k]_0$ as follows. $\Psi_0^j(S_0) = 0$, for all $j \in [k]_0$. For each $i \in [a]$, $\Psi_i^0(S_i) = 0$ and $\Psi_i^j(S_i) = \Phi_i$, for all $j \in [k]$. For each $i \in \{a+1, \ldots, z\}$ and $j \in [k]_0$, $\Psi_i^j(S_i) = j$. Based on these coloring functions for the sets in the tuple, we now define coloring functions for the hyperedges of H. For that, let us first classify the hyperedges of H based on the sets in the tuple $\mathbf{t}$. For each $i \in \{0, a+1, a+2, \ldots, z\}$, let $E_{S_i} = \{F \in E(H) : F \subseteq S_i\}$. For each $i \in [a]$, let $E_{S_i} = \{F \in E(H) : F \cap S_i \neq \emptyset\}$. Since $\mathbf{t}$ is a sets-colorings tuple and $S_0, S_1, \ldots, S_z$ are the sets of this tuple, renamed as described above, $E(H) = \uplus_{i \in [z]_0} E_{S_i}$. We now define the coloring functions for the hyperedges of H. For each $i \in \{0, a+1, \ldots, z\}$, $F \in E_{S_i}$ and $j \in [k]_0$, $\Psi_F^j = \Psi_i^j|_F$. For each $i \in [a]$, for each $F \in E_{S_i}$ and $j \in [k]_0$, $\Psi_F^j(v) = \Psi_i^j(v)$, if $v \in S_i$; $\Psi_F^j(v) = 0$, if $v \in S_0$.

This finishes the description of the coloring functions for the sets of the tuples and the hyperedges of H. Observe that the colorings Ψ defined for hyperedges are consistent with $V(H)$, that is, for each $F \in E(H)$, no matter which coloring out of Ψ_F^j, $j \in [z]_0$ is picked, it together colors $V(H)$ where each vertex in $V(H)$ gets a unique color (assuming $V(H) = \cup_{F \in E(H)} V(F)$, where $V(F)$ denotes the vertices in the hyperedge F). This is true because $\mathbf{t}$ is a sets-colorings tuple and $\Psi_{S_0} = 0$.

For each $i \in [z]$, let $E_{S_i} = \{F_{i,1}, \ldots, F_{i,z_i}\}$. Fix a set S_i and $j \in [k]_0$, and define

$$h_i^j[\mu', l_1', l_2'] = \bigvee_{\substack{(\mu^r)_{r \in [z_i]} \\ (l_1^r)_{r \in [z_i]} \\ (l_2^r)_{r \in [z_i]}}} \bigwedge_{r \in [z_i]} f_{F_{i,r}}(\Psi^j_{F_{i,r}}, \mu^r, {l_1}^r, {l_2}^r)$$

where $\sum_{r\in[z_i]} \mu^r = \mu'$, $\sum_{r\in[z_i]} l_1^r \leq l_1'$, $\sum_{r\in[z_i]} {l_2}^r \leq l_2'$, and each μ^r, l_1^r, l_2^r is a non-negative integer.

Now define $\mathcal{H}[i,\mu',l_1',l_2'] = \bigvee_{j\in[k]_0} h_i^j[\mu',l_1',l_2']$.

Let

$$\texttt{computeHP}[\mu,l_1,l_2] = \bigvee_{\substack{(\mu^i)_{i\in[z]_0}\\(l_1^i)_{i\in[z]_0}\\(l_2^i)_{i\in[z]_0}}} \bigwedge_{i\in[z]_0} \mathcal{H}[i,\mu^i,l_1^i,l_2^i]$$

where $\sum_{i\in[z]_0} \mu^i = \mu$, $\sum_{i\in[z]_0} l_1^i \leq l_1$, $\sum_{i\in[z]_0} l_2^i \leq l_2$, and each μ^i, l_1^i, l_2^i is a non-negative integer. Note that each of the functions $\Psi_i^j, \Psi_F^j, h_i^j, \mathcal{H}$ and `computeHP` are defined with respect to a sets-colorings tuple $\mathbf{t}$.

Lemma 12.5.17. *Suppose* $\texttt{aHP}[\mu,l_1,l_2] = 1$. *Then there exists a sets-colorings tuple* $\boldsymbol{t} \in \mathcal{T}$, *such that, for this tuple* $\boldsymbol{t}$, $\texttt{computeHP}[\mu,l_1,l_2] = 1$.

Proof. Recall that

$$\texttt{aHP}[\mu,l_1,l_2] = \bigvee_{\Upsilon:V(H)\to[k]_0} \bigvee_{\substack{(\mu^F)|_{F\in E(H)}\\(l_1^F)|_{F\in E(H)}\\(l_2^F)|_{F\in E(H)}}} \bigwedge_{F\in E(H)} f_F(\Upsilon|_F,\mu^F,l_1^F,l_2^F),$$

where $\sum_{F\in E(H)} \mu^F = \mu$, $\sum_{F\in E(H)} l_1^F \leq l_1$, $\sum_{F\in E(H)} l_2^F \leq l_2$ and each of μ^F, l_1^F and l_2^F is a non-negative integer.

Since $E(H) = \uplus_{i\in[z]_0} E_{S_i}$, we have the following.

$$\texttt{aHP}[\mu,l_1,l_2] = \bigvee_{\substack{\Upsilon:V(H)\to[k]_0\\(\mu^F)_{F\in E(H)}\\(l_1^F)_{F\in E(H)}\\(l_2^F)_{F\in E(H)}}} \bigwedge_{i\in[z]_0} \bigwedge_{F\in E_{S_i}} f_F(\Upsilon|_F,\mu^F,l_1^F,l_2^F),$$

where $\sum_{F\in E(H)} \mu^F = \mu$, $\sum_{F\in E(H)} l_1^F \leq l_1$, $\sum_{F\in E(H)} l_2^F \leq l_2$, and for all $F \in E(H)$, μ^F, l_1^F and l_2^F are non-negative integers.

Since $\texttt{aHP}[\mu,l_1,l_2] = 1$, there exists a witnessing assignment $\Upsilon : V(H) \to [k]_0$. Since I is a favorable instance of HP, Υ clearly satisfies the local unbreakability and global unbreakability conditions of Lemma 12.5.1 (when the input to the algorithm of Lemma 12.5.1 is $(H,k,d,k,3k^2,q)$). We first show that, Υ also satisfies the first

precondition of Lemma 12.5.1 with $x = k$. That is, the number of hyperedges of H that are non-monochromatic under Υ is at most k. Since $\texttt{aHP}[\mu, l_1, l_2] = 1$, for all $F \in E(H)$ there exist μ^F, l_1^F and l_2^F such that $f_F(\Upsilon_{|F}, \mu^F, l_1^F, l_2^F) = 1$. Hence, the connectivity property of Υ (which exists because I is a favorable instance) implies that for each F that is not monochromatic in Υ, we have that $l_1^F + l_2^F \geq 1$. However, $\sum_{F \in E(H)} l_1^F + l_2^F \leq l_1 + l_2 \leq k_1 + k_2 = k$. Thus, the number of non-monochromatic hyperedges under Υ is at most k.

Thus, from Lemma 12.5.1, there exists a good tuple $\mathbf{t} \in \mathcal{T}$, for Υ. Consider $\texttt{computeHP}[\mu, l_1, l_2]$ (and thus corresponding $h_i^j, \Psi_F^j, \Psi_i^j$) defined for this good tuple. From the definition of a good tuple and Ψ_i^j, for each $i \in [z]_0$, $\Upsilon|_{S_i} = \Psi_i^j$, for some $j \in [k]_0$. Also, since $\mathbf{t}$ is a sets-colorings tuple, for each $F \in E(H)$, $\Upsilon|_F = \Psi_F^j$, for some $j \in [k]_0$.

Thus, if $\texttt{aHP}[\mu, l_1, l_2] = 1$, we get the following.

$$\texttt{aHP}[\mu, l_1, l_2] = \bigvee_{\substack{(\mu^i)_{i\in[z]_0} \\ (l_1^i)_{i\in[z]_0} \\ (l_2^i)_{i\in[z]_0}}} \bigwedge_{i\in[z]_0} \mathcal{H}[i, \mu^i, l_1^i, l_2^i] = \texttt{computeHP}[\mu, l_1, l_2],$$

where $\sum_{i\in[z]_0} \mu^i = \mu$, $\sum_{i\in[z]_0} l_1^i \leq l_1$, $\sum_{i\in[z]_0} l_2^i \leq l_2$, and each μ^i, l_1^i, l_2^i is a non-negative integer. □

Lemma 12.5.18. *If, there exists a tuple $\boldsymbol{t} \in \mathcal{T}$, such that, for this tuple $\boldsymbol{t}$, $\texttt{computeHP}[\mu, l_1, l_2] = 1$, then $\texttt{aHP}[\mu, l_1, l_2] = 1$.*

Proof. From the definition of computeHP, we have the following.

$$\texttt{computeHP}[\mu, l_1, l_2] = \bigvee_{\substack{(\mu^i)_{i\in[z]_0} \\ (l_1^i)_{i\in[z]_0} \\ (l_2^i)_{i\in[z]_0}}} \bigwedge_{i\in[z]_0} \mathcal{H}[i, \mu^i, l_1^i, l_2^i] = 1,$$

where $\sum_{i\in[z]_0} \mu^i = \mu$, $\sum_{i\in[z]_0} l_1^i \leq l_1$, $\sum_{i\in[z]_0} l_2^i \leq l_2$, and each μ^i, l_1^i, l_2^i is a non-negative integer.

Since $\mathcal{H}[i, \mu', l_1', l_2'] = \bigvee_{j\in[k]_0} h_i^j[\mu', l_1', l_2']$, we get the following for some $j_0, \ldots, j_z \in [k]_0$.

$$\bigvee_{\substack{(\mu^i)_{i\in[z]_0}\\(l_1^i)_{i\in[z]_0}\\(l_2^i)_{i\in[z]_0}}} \bigwedge_{i\in[z]_0} h_i^{j_i}[\mu^i, l_1^i, l_2^i] = 1$$

From the definition of $h_i^{j_i}$, we get the following.

$$\bigvee_{\substack{(\mu^{F_{i,r}})|_{i\in[z]_0,r\in[z_i]}\\(l_1^{F_{i,r}})|_{i\in[z]_0,r\in[z_i]}\\(l_2^{F_{i,r}})|_{i\in[z]_0,r\in[z_i]}}} \bigwedge_{i\in[z]_0}\bigwedge_{r\in[z_i]} f_{F_{i,r}}(\Psi_{F_{i,r}}^{j_i}, \mu^{F_{i,r}}, l_1^{F_{i,r}}, l_2^{F_{i,r}}) = 1,$$

where $\sum_{i\in[z]_0,r\in[z_i]} \mu^{F_{i,r}} = \mu$, $\sum_{i\in[z]_0,r\in[z_i]} l_1^{F_{i,r}} \leq l_1$, $\sum_{i\in[z]_0,r\in[z_i]} l_2^{F_{i,r}} \leq l_2$ and each of μ^F, l_1^F and l_2^F is a non-negative integer.

Since Ψ for hyperedges gives a consistent coloring for $V(H)$, let this coloring be $\Upsilon : V(H) \to [k]_0$. This coloring Υ then witnesses the following.

$$\bigvee_{\substack{(\mu^{F_{i,r}})|_{i\in[z]_0,r\in[z_i]}\\(l_1^{F_{i,r}})|_{i\in[z]_0,r\in[z_i]}\\(l_2^{F_{i,r}})|_{i\in[z]_0,r\in[z_i]}}} \bigwedge_{i\in[z]_0}\bigwedge_{r\in[z_i]} f_{F_{i,r}}(\Upsilon|_{F_{i,r}}, \mu^{F_{i,r}}, l_1^{F_{i,r}}, l_2^{F_{i,r}}) = 1$$

where $\sum_{i\in[z]_0,r\in[z_i]} \mu^{F_{i,r}} = \mu$, $\sum_{i\in[z]_0,r\in[z_i]} l_1^{F_{i,r}} \leq l_1$, $\sum_{i\in[z]_0,r\in[z_i]} l_2^{F_{i,r}} \leq l_2$ and each of μ^F, l_1^F and l_2^F is a non-negative integer.

Since $E(H) = \uplus_{i\in[z]_0} E_{S_i}$, we get the following.

$$\bigvee_{\substack{(\mu^F)|_{F\in E(H)}\\(l_1^F)|_{F\in E(H)}\\(l_2^F)|_{F\in E(H)}}} \bigwedge_{F\in E(H)} f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F) = 1,$$

where $\sum_{F\in E(H)} \mu^F = \mu$, $\sum_{F\in E(H)} l_1^F \leq l_1$, $\sum_{F\in E(H)} l_2^F \leq l_2$, and for all $F \in E(H)$, μ^F, l_1^F and l_2^F are non-negative integers.

Thus, we conclude that $\texttt{aHP}[\mu, l_1, l_2] = 1$. □

From Lemma 12.5.17 and Lemma 12.5.18, we conclude that, to compute $\texttt{aHP}[\mu, l_1, l_2]$, it is enough to compute $\texttt{computeHP}[\mu, l_1, l_2]$ for each tuple $\mathbf{t} \in \mathcal{T}$. In the upcom-

ing lemmas we analyse the time taken to compute $\texttt{computeHP}[\mu, l_1, l_2]$ for any tuple $\mathbf{t} \in \mathcal{T}$.

Lemma 12.5.19. *For any $i \in [z]_0, j \in [k]_0, \mu' \leq \mu, l_1' \leq l_1 \leq k_1, l_2' \leq l_2 \leq k_2$, $h_i^j[\mu', l_1', l_2']$ can be computed in time $\mathcal{O}(z_i \cdot (\mu \cdot k_1 \cdot k_2)^2)$.*

Proof. Recall

$$h_i^j[\mu', l_1', l_2'] = \bigvee_{\substack{(\mu^r)_{r\in[z_i]} \\ (l_1^r)_{r\in[z_i]} \\ (l_2^r)_{r\in[z_i]}}} \bigwedge_{r\in[z_i]} f_{F_{i,r}}(\Psi^j_{F_{i,r}}, \mu^r, {l_1}^r, {l_2}^r)$$

where $\sum_{r\in[z_i]} \mu^r = \mu'$, $\sum_{r\in[z_i]} l_1^r \leq l_1'$, $\sum_{r\in[z_i]} {l_2}^r \leq l_2'$, and each μ^r, l_1^r, l_2^r is a non-negative integer.

For any $c \in [z_i], \mu', l_1'$ and l_2', let

$$h_i^j[c, \mu', l_1', l_2'] = \bigvee_{\substack{(\mu^r)_{r\in[c]} \\ (l_1^r)_{r\in[c]} \\ (l_2^r)_{r\in[c]}}} \bigwedge_{r\in[c]} f_{F_{i,r}}(\Psi^j_{F_{i,r}}, \mu^r, {l_1}^r, {l_2}^r)$$

where $\sum_{r\in[c]} \mu^r = \mu'$, $\sum_{r\in[c]} l_1^r \leq l_1'$, $\sum_{r\in[c]} {l_2}^r \leq l_2'$, and each μ^r, l_1^r, l_2^r is a non-negative integer.

Then $h_i^j[z_i, \mu', l_1', l_2']$ can be computed using the following recurrences in time $\mathcal{O}(z_i \cdot (\mu \cdot k_1 \cdot k_2)^2)$.

$$h_i^j[1, \mu', l_1', l_2'] = f_{F_{i,1}}(\Psi^j_{F_{i,1}}, \mu', l_1', l_2')$$

For all $c \in \{2 \ldots, z_i\}$,

$$h_i^j[c, \mu', l_1', l_2'] = \bigvee_{\substack{\mu'=\mu^1+\mu^2 \\ l_1'\geq l_1^1+l_1^2 \\ l_2'\geq l_2^1+l_2^2}} h_i^j[c-1, \mu^1, l_1^1, l_2^1] \wedge f_{F_{i,c}}(\Psi^j_{F_{i,c}}, \mu^2, l_1^2, l_2^2)$$

Observe that $h_i^j[\mu', l_1', l_2'] = h_I^j[z_i, \mu', l_1', l_2']$. This concludes the proof. □

Since $\mathcal{H}[i, \mu', l_1', l_2'] = \bigvee_{j\in[k]_0} h_i^j[\mu', l_1', l_2']$, from Lemma 12.5.19, for any i, μ', l_1', l_2',

$\mathcal{H}[i, \mu', l_1', l_2']$ can be computed in time $\mathcal{O}(z_i \cdot (\mu)^2 \cdot (k_1 + k_2)^3)$.

Lemma 12.5.20. *For any tuple $\boldsymbol{t} \in \mathcal{T}$, for any $\mu' \leq \mu, l_1' \leq l_1 \leq k_1$, $l_2' \leq l_2 \leq k_2$, $\texttt{computeHP}[\mu', l_1', {}_2']$ can be computed in time $z \cdot z_i \cdot (\mu \cdot (k_1 + k_2)^{\mathcal{O}(1)})$.*

Proof. Recall

$$\texttt{computeHP}[\mu, l_1, l_2] = \bigvee_{\substack{(\mu^i)_{i \in [z]_0} \\ (l_1^i)_{i \in [z]_0} \\ (l_2^i)_{i \in [z]_0}}} \bigwedge_{i \in [z]_0} \mathcal{H}[i, \mu^i, l_1^i, l_2^i],$$

where $\sum_{i \in [z]_0} \mu^i = \mu$, $\sum_{i \in [z]_0} l_1^i \leq l_1$, $\sum_{i \in [z]_0} l_2^i \leq l_2$, and each μ^i, l_1^i, l_2^i is a non-negative integer.

For any $c \in [z]_0, \mu', l_1'$ and l_2', let

$$\texttt{computeHP}[c, \mu, l_1, l_2] = \bigvee_{\substack{(\mu^i)_{i \in [c]_0} \\ (l_1^i)_{i \in [c]_0} \\ (l_2^i)_{i \in [c]_0}}} \bigwedge_{i \in [c]_0} \mathcal{H}[i, \mu^i, l_1^i, l_2^i],$$

where $\sum_{i \in [c]} \mu^i = \mu$, $\sum_{i \in [c]} l_1^i \leq l_1$, $\sum_{i \in [c]} l_2^i \leq l_2$, and each μ^i, l_1^i, l_2^i is a non-negative integer.

Then $\texttt{computeHP}[z, \mu, l_1, l_2]$ can be computed using the following recurrences.

$$\texttt{computeHP}[0, \mu, l_1, l_2] = \mathcal{H}[0, \mu, l_1, l_2]$$

For all $c \in [z]$,

$$\texttt{computeHP}[c, \mu, l_1, l_2] = \bigvee_{\substack{\mu' = \mu^1 + \mu^2 \\ l_1' \geq l_1^1 + l_1^2 \\ l_2' \geq l_2^1 + l_2^2}} \texttt{computeHP}[c-1, \mu^1, l_1^1, l_2^1] \wedge \mathcal{H}[c, \mu^2, l_1^2, l_2^2]$$

Observe that $\texttt{computeHP}[\mu, l_1, l_2] = \texttt{computeHP}[z, \mu, l_1, l_2]$. From Lemma 12.5.19, for any i, μ, l_1, l_2, $\mathcal{H}[i, \mu, l_1, l_2]$ can be solved in time $\mathcal{O}(z_i \cdot (\mu)^2) \cdot (k_1 + k_2)^3$. Thus, $\texttt{computeHP}[\mu, l_1, l_2]$ can be solved in time $z \cdot z_i \cdot (\mu \cdot (k_1 + k_2)^{\mathcal{O}(1)})$. This concludes the proof. □

From Lemma 12.5.1, the number of tuples in $\mathcal{T}$ is bounded and for each tuple, the time taken to compute `computeHP`$[\mu, l_1, l_2]$ is given by Lemma 12.5.20. Thus, Lemma 12.5.1 and Lemma 12.5.20 together give the desired running time bound of Theorem 12.4.2.

Chapter 13

Degeneracy Reduction Preserving Minimal Multicuts

We begin by defining the MULTICUT problem.

MULTICUT
Input: An (undirected) graph G, a set $T = \{\{s_1, t_1\}, \{s_2, t_2\}, \ldots, \{s_\ell, t_\ell\}\}$ of terminal pairs and an integer k.
Question: Does there exist $S \subseteq V(D)$ of size at most k such that for every $i \leq \ell$, s_i and t_i are in distinct connected components of $G - S$?

Such a set S is called a *multicut* of T in G. In this chapter, we show that all the minimal multicuts of any graph G are, confined inside a subgraph of bounded degeneracy. In fact, such a subgraph can be found in polynomial time. More specifically, we prove the following theorem.

Theorem 13.0.1 (Degeneracy Reduction Theorem)**.** *There exists a polynomial time algorithm that given a graph G, a set $T = \{\{s_1, t_1\}, \{s_2, t_2\}, \ldots, \{s_\ell, t_\ell\}\}$ of terminal pairs and an integer k, returns an induced subgraph $G^\star$ of G and a subset $T^\star$ of T which have the following properties.*

- *every minimal multicut of T in G of size at most k is a minimal multicut of $T^\star$ in $G^\star$,*
- *every minimal multicut of $T^\star$ in $G^\star$ of size at most k is a minimal multicut of T in G, and*
- *$G^\star$ does not contain a $(k+2)$-connected set of size $\mathcal{O}(64^k \cdot k^2)$.*

We remark that excluding a $(k+2)$-connected set of size $\mathcal{O}(64^k \cdot k^2)$ implies that $G^\star$ excludes a clique of size $\mathcal{O}(64^k \cdot k^2)$ as a *topological minor* (and hence, the degeneracy of $G^\star$ is bounded by $2^{\mathcal{O}(k^{\mathcal{O}(1)})}$). In fact, the property of excluding a large $(k+2)$-connected set puts considerable extra restrictions on the graph, on top of being topological minor free, as there exist planar graphs that contain arbitrarily large $(k+2)$-connected sets. The proof of Theorem 13.0.1 uses the *irrelevant vertex* technique of Robertson and Seymour [188], however, instead of topological arguments for finding an irrelevant vertex we rely on a careful case distinction based on cut-flow duality together with counting arguments based on *important separators*.

13.1 Outline of the Proof

Towards the proof of Theorem 13.0.1, we describe an algorithm that given G, the set T of terminal pairs, an integer k and a $(k+2)$-connected set W of size at least $64^{k+2} \cdot (k+2)^2$, computes a vertex v that does not appear in any minimal multicut of size at most $k+1$. One can show that such a vertex v is *irrelevant* in the sense that G, T has *exactly* the same family of minimal multicuts of size at most k as the graph $G - v$ with the terminal set $T' = \{\{s_i, t_i\} \in T : v \notin \{s_i, t_i\}\}$. The proof of Theorem 13.0.1 then follows by repeatedly removing irrelevant vertices, until $|W| \leq 64^{k+2} \cdot (k+2)^2$.

Degree 1 Terminals Assumption. In order to identify an irrelevant vertex it is helpful to assume that every terminal s_i or t_i has degree 1 in G, and that no vertex in G appears in more than one terminal pair. To justify this assumption one can, for every pair $\{s_i, t_i\} \in T$, add $k+2$ new degree 1 vertices $s_i^1, s_i^2, \ldots, s_i^{k+2}$ and make them adjacent to s_i, and $k+2$ new degree 1 vertices $t_i^1, t_i^2, \ldots, t_i^{k+2}$ and make them adjacent to t_i. Call the resulting graph G', and make a terminal pair set T' from T by inserting for every pair $\{s_i, t_i\} \in T$ the set $\{\{s_i^j, t_i^j\} : 1 \leq j \leq k+2\}$ into T'. It is clear that the set of (minimal) multicuts of T' in G' of size at most $k+1$ is the same as the set of (minimal) multicuts of T in G of size at most $k+1$.

Detecting Irrelevant Vertices. In order to identify an irrelevant vertex we investigate the properties of all vertices $v \in W$ for which there exists a minimal multicut of size at most $k+1$ containing v. We will call such vertices *relevant*. Let $v \in W$ be a relevant vertex and let S be a minimal multicut of size at most $k+1$ containing v. Since W is a $(k+2)$-connected set and $|S| \leq k+1$, $W \setminus S$ is contained in some connected component C of $G - S$. Since S is a multicut we also have that S is a *pair*

cut for T with respect to W in the following sense: for each terminal pair $\{s_i, t_i\}$ at most one of s_i and t_i can reach $W \setminus S$ in $G - S$. This is true because all vertices of $W \setminus S$ lies in the same connected component of $G - S$. Furthermore, $S \setminus \{v\}$ *cannot* be a pair cut for T with respect to W, because if it happened to be a pair cut, then we can show that $S \setminus \{v\}$ would also have been a multicut, contradicting the minimality of S. We say that $v \in W$ is *essential* if there exists some pair cut S for T with respect to W such that $|S| \leq k+1$, $v \in S$ and $S \setminus \{v\}$ is not a pair cut for T with respect to W. The above argument shows that every relevant vertex is essential, and it remains to find a vertex $v \in W$ which is provably not essential.

The algorithm that searches for a non-essential vertex v crucially exploits *important separators*, defined by Marx [156]. Given a graph G and two vertex sets A and B, an *A-B-separator* is a vertex set $S \subseteq V(G)$ such that there is no path from $A \setminus S$ to $B \setminus S$ in $G - S$. An A-B-separator S is called a *minimal A-B-separator* if no proper subset of S is also an A-B-separator. Given a vertex set S, we define the *reach of A in $G - S$* as the set $R_G(A, S)$ of vertices reachable from A by a path in $G - S$. We can now define a partial order on the set of minimal A-B separators as follows. Given two minimal A-B separators S_1 and S_2, we say that S_1 is "at least as good as" S_2 if $|S_1| \leq |S_2|$ and $R_G(A, S_2) \subsetneq R_G(A, S_1)$. In plain words, S_1 "costs less" than S_2 in terms of the number of vertices deleted and S_1 "is pushed further towards B" than S_2 is. A minimal A-B-separator S is an *important A-B-separator* if no minimal A-B-separator other than S is at least as good as S. A key insight behind many parameterized algorithms [51, 54, 57, 72, 135, 141, 145, 148, 147, 160] is that for every k, the number of important A-B-separators of size at most k is at most 4^k [50]. We refer the reader to Marx [156] and the textbook by Cygan et al. [65] for a more thorough exposition of important separators.

The algorithm that searches for a non-essential vertex v makes the following case distinction. Either there exists a small T-W-separator Z, or there are many vertex disjoint paths from T to W. Here, we have abused notation by treating T as a set of vertices in the terminal pairs rather than a set of terminal pairs. As pointed out by an anonymous reviewer, in both the cases, the essence of the arguments is to set up the stage for the application of the anti-isolation lemma of Marx [157], which appeared in [178], to mark relevant vertices and relevant terminal pairs respectively. In the first case, when there exists a T-W-separator Z of size at most $\zeta = 16^{k+1} \cdot 64(k+2)$, we show that every relevant vertex $v \in W$ is contained in some important z-W-separator of size at most $k+1$, for some $z \in Z$. Since there are at most 4^{k+1} such important separators and we can enumerate them efficiently [50],

the algorithm simply marks all the vertices in W appearing in such an important separator and outputs one vertex that is not marked.

Many Disjoint Paths. If there are at least $16^{k+1} \cdot 64(k+2)$ vertex disjoint paths from T to W we identify a terminal pair $\{s_i, t_i\}$ such that, for every minimal multicut S of size at most $k+1$ for the instance G with terminal set $T \setminus \{\{s_i, t_i\}\}$, S is also a minimal multicut for G with terminal set T. Such a terminal pair is irrelevant in the sense that removing $\{s_i, t_i\}$ from T does not change the family of minimal multicuts of size at most $k+1$. Thus, if we later identify a vertex $v \in W$ that is irrelevant with the reduced terminal set, then v is also irrelevant with respect to the original terminal set. We will say that a terminal pair that isn't irrelevant is *relevant*.

To identify an irrelevant terminal pair we proceed as follows. Without loss of generality, there are $\zeta/2$ vertex disjoint paths from $A = \{s_1, s_2, \ldots s_{\zeta/2}\}$ to W. Thus, for any set S of at most $k+2$ vertices, all of A except for at most $k+2$ vertices can reach $W \setminus S$ in $G - S$. Let $B = \{t_1, t_2, \ldots t_{\zeta/2}\}$. We have that for every pair cut S for T with respect to W, at most $k+2$ vertices of $B \setminus S$ are reachable from W in $G - S$.

Consider a pair $\{s_i, t_i\}$ with $s_i \in A$ and $t_i \in B$. If $\{s_i, t_i\}$ is relevant, then there must exist a set S of size at most $k+1$ that is a minimal pair cut for G with terminals $T \setminus \{\{s_i, t_i\}\}$ with respect to W, but is not a pair cut with terminal pair set T. We have that t_i is reachable from $W \setminus S$ in $G - S$, and that $S \cup \{t_i\}$ is a pair cut for T. Let $\hat{B} \subseteq B$ be the set of vertices in B that are reachable from W in $G - (S \cup \{t_i\})$. From the discussion in the previous paragraph it follows that $|\hat{B}| \leq k+2$. Thus, $S \cup \{t_i\} \cup \hat{B}$ is a W-B separator of size at most $2(k+2)$. Pick any minimal W-B separator $\hat{S} \subseteq S \cup \{t_i\} \cup \hat{B}$.

We argue that $t_i \in \hat{S}$. To that end we show that there exists a path P from W to t_i in $G - (S \cup \hat{B})$. Thus, if $t_i \notin \hat{S}$ then $\hat{S}$ would be a subset of $S \cup \hat{B}$ and P would be a path from W to B in $G - \hat{S}$, contradicting that $\hat{S}$ is a W-B-separator. We know that there exists a path P from W to t_i in $G - S$ and that P does not visit any vertex in $\hat{B}$ on the way to t_i because all vertices in $\hat{B}$ have degree 1. Hence P is disjoint from $\hat{S}$, yielding the desired contradiction. We conclude that $t_i \in \hat{S}$.

With all of this hard work we have, under the assumption that $\{s_i, t_i\}$ is a relevant pair with $t_i \in B$, exhibited a minimal W-B-separator $\hat{S}$ that contains t_i. There must exist some important W-B-separator $S^\star$ that is at least as good as $\hat{S}$. Since all the vertices of P (except t_i) are reachable from W in $G - \hat{S}$ it follows that $t_i \in S^\star$. We have now shown that if $\{s_i, t_i\}$ is a relevant pair with $t_i \in B$, then there exists a W-

B important separator of size at most $2(k+2)$ that contains t_i. The algorithm goes over all W-B important separators of size at most $2(k+2)$ and marks all vertices appearing in such important separators. Since $\zeta/2 > 4^{2(k+2)} \cdot 2(k+2)$ it follows that some vertex t_i in B is left unmarked. The pair (s_i, t_i) is then an irrelevant pair. This concludes the proof sketch that there exists a polynomial time algorithm that given G, T, k and W finds an irrelevant vertex in W, provided that W is large enough. We would like to remark here (as pointed out by an anonymous reviewer) that this process is similar in principle to the anti-isolation lemma of Marx [157] (which also appeared in [178]).

Finding a Large $(k+2)$-Connected Set. We have shown how to identify an irrelevant vertex given a $(k+2)$-connected set W of large size. But how to find such a set W, if it exists? Given G we can in polynomial time build an auxiliary graph $G^\star$ that has the same vertex set as G. Two vertices in $G^\star$ are adjacent if there are at least $k+2$ internally vertex disjoint paths between them in G. Clearly $(k+2)$-connected sets in G are cliques in $G^\star$ and vice versa. However, finding cliques in general graphs is $W[1]$-hard, and is believed to not have an approximation even in FPT time. To get around this obstacle we exploit the special structure of $G^\star$.

A $(k+2)$-connected set W in G of size at least $64^{k+2} \cdot 4(k+2)^2$ induces a subgraph of $G^\star$ where every vertex has degree at least $(k+2)$. Thus the degeneracy of $G^\star$ is at least $64^{k+2} \cdot 4(k+2)^2$. A modification of a classic result of Mader [152] (see also Diestel [76] and lecture notes of Sudakov [199]) shows that every graph of degeneracy at least $4d$ contains a $(d+1)$-connected set of size at least $d+2$, and that such a set can be computed in polynomial time. We apply this result with $d = 64^{k+2} \cdot (k+2)^2 - 1$ to obtain a $(64^{k+2} \cdot (k+2)^2)$-connected set in $W^\star$ in $G^\star$ of size at least $64^{k+2} \cdot (k+2)^2$. A simple argument shows that $W^\star$ is also a $(k+2)$-connected set in G. We may now apply the algorithm to detect irrelevant vertices using $W^\star$. This concludes the proof sketch of Theorem 13.0.1.

The remaining chapter is organized as follows. We start by showing how to efficiently find some vertices that are irrelevant to "small" digraph pair cuts (defined in Section 13.2), assuming that the input graph has a sufficiently large number of vertices that are in-neighbors of the root. Afterwards, having a method to identify such irrelevant vertices at hand, we develop (in Section 13.3) an efficient algorithm that given a graph G, a set of terminal pairs T and a positive integer k, outputs an induced subgraph $G^\star$ of G and a subset $T^\star \subseteq T$ such that the following conditions are satisfied. First, any set $S \subseteq V(G)$ of size at most k is a minimal multicut of T in G if and only if $S \subseteq V(G^\star)$ and it is a minimal multicut of $T^\star$ in $G^\star$. Second, $G^\star$

does not contain any "large" $(k+2)$-connected set.

13.2 Identifying Vertices Irrelevant to Digraph Pair Cuts

The notion of a digraph pair cut was defined by Kratsch and Wahlström in [136]. This notion was used to derive randomized polynomial kernels for many problems, including Almost 2-SAT and Multiway Cut with Deletable Terminals. Towards defining which vertices are irrelevant to "small" digraph pair cuts, we first formally define what is a digraph pair cut.

Definition 13.2.1. *Let D be a digraph, T be a set of pairs of vertices (called* terminal pairs*), and $r \in V(D)$. We say that $S \subseteq V(D) \setminus \{r\}$ is an T-r-*digraph pair cut *if for every terminal pair $\{s,t\} \in T$, S is an s-r-separator or a t-r-separator.*[1]

The problem Digraph Pair Cut takes as input a digraph D, a set of terminal pairs T, $r \in V(D)$ and $k \in \mathbb{N}$, and the task is to output Yes if and only if there is an T-r-digraph pair cut in G of size at most k. We say that a vertex $v \in V(D)$ is *irrelevant* to the instance (D,T,r,k) if there is no minimal T-r-digraph pair cut of size at most k in D that contains v. If a vertex is not irrelevant to (D,T,r,k), then we say that it is *relevant* to (D,T,r,k). In the following lemma, which is the main result of this subsection, we show that for an instance (D,T,r,k) of Digraph Pair Cut, the number of in-neighbors of r that belong to at least one minimal T-r-digraph pair cut of size at most k is upper bounded by $64^{k+1}(k+1)^2$. In other words, we bound the number of in-neighbors of r that are relevant.

Lemma 13.2.1. *Let (D,T,r,k) be an instance of* Digraph Pair Cut*. The number of vertices in $N_D^-(r)$ that are relevant to (D,T,r,k) is at most $64^{k+1}(k+1)^2$. Moreover, there is a deterministic algorithm that given (D,T,r,k), runs in time $\mathcal{O}(|T| \cdot n(n^{\frac{2}{3}}+m))$, and outputs a set $R \subseteq N_D^-(r)$ of size at most $64^{k+1}(k+1)^2$ which contains all relevant vertices to (D,T,r,k) in $N_D^-(r)$.* [2]

Towards the proof of Lemma 13.2.1, we first define which terminal pairs are irrelevant.

[1] The definition of digraph pair cut used here is same as that of Kratsch and Wahlström [136] where we reverse the directions of the arcs of the graph.

[2] In other words, the vertices in $N_D^-(r) \setminus R$ are irrelevant to (D,T,r,k).

Definition 13.2.2. *Let (D, T, r, k) be an instance of* Digraph Pair Cut. *A terminal pair $\{s, t\} \in T$ is* irrelevant *to (D, T, r, k) if any minimal $(T \setminus \{\{s, t\}\})$-$r$-digraph pair cut in D of size at most k is also a minimal T-r-digraph pair cut in D.*

The following observation directly follows from the definition of irrelevant terminal pairs.

Observation 13.2.1. *Let D be a digraph, T be a set of terminal pairs, $r \in V(D)$ and $k \in \mathbb{N}$. If $\{s, t\} \in T$ is a terminal pair irrelevant to (D, T, r, k), then any vertex relevant to (D, T, r, k) is also a vertex relevant to $(D, T \setminus \{\{s, t\}\}, r, k)$.*

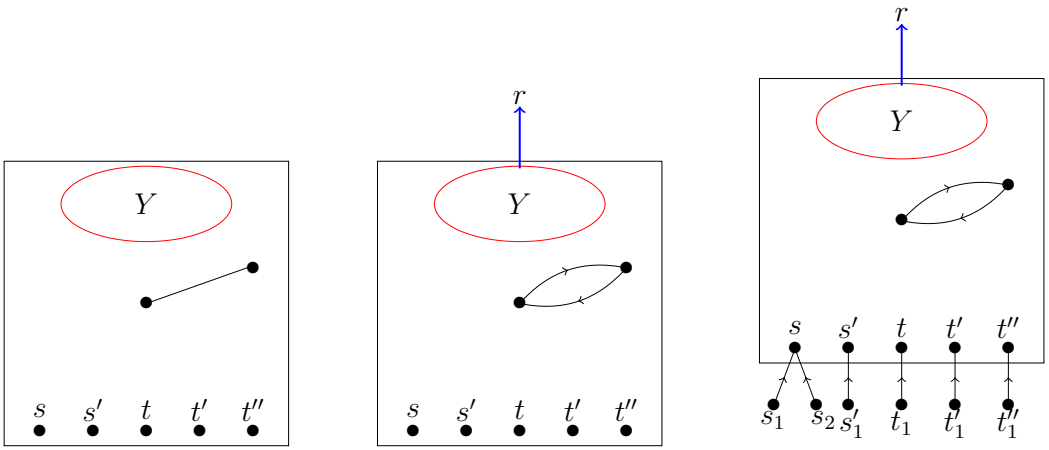
We now define *important separators*, which have played an important role in the context of existing literature concerning cut related problems.

Definition 13.2.3 (Important Separators, [156])**.** *Let D be a digraph. For subsets $X, Y, S \subseteq V(D)$, the set of vertices reachable from $X \setminus S$ in $D - S$ is denoted by $R_D(X, S)$. An X-Y-separator S* dominates *an X-Y-separator S' if $|S| \leq |S'|$ and $R_D(X, S') \subsetneq R_D(X, S)$. A subset S is an* important X-Y-separator *if it is minimal, and there is no X-Y-separator S' that dominates S. For two vertices $s, t \in V(D)$, the term* important s-t-separator *refers to an important $N_D^+(s)$-$N_D^-(t)$-separator in $D - \{s, t\}$. For $r \in V(D)$ and $Y \subseteq V(D)$, the term* important Y-r-separator *refers to an important Y-$N_D^+(r)$-separator in $D - r$.*

Lemma 13.2.2 ([50, 156])**.** *Let D be a digraph, $X, Y \subseteq V(D)$, and $k \in \mathbb{N}$. The number of important X-Y-separators of size at most k is upper bounded by 4^k, and these separators can be enumerated in time $\mathcal{O}(4^k \cdot k \cdot (n + m))$.*

The rest of this subsection is dedicated to the proof of Lemma 13.2.1. That is, we design an algorithm, called $\mathcal{A}$, that finds a set R with the properties specified by Lemma 13.2.1. If $|N_D^-(r)| \leq 64^{k+1}(k+1)^2$, then $N_D^-(r)$ is the required set R. Thus, from now on, we assume that $|N_D^-(r)| > 64^{k+1}(k+1)^2$. Algorithm $\mathcal{A}$ is an iterative algorithm. In each iteration, $\mathcal{A}$ either terminates by outputting the required set R, or finds an irrelevant terminal pair for the input instance, removes it from the set of terminal pairs, and then repeats the process.

As a preprocessing step preceding the first call to $\mathcal{A}$, we modify the graph D and the set of terminal pairs T as described below. The new graph D' and set of terminal pairs T' would allow us to accomplish our task while simplifying some arguments in the proof. We set D' to be the digraph obtained from D by adding two new vertices,



(a) Let Y be a $(k+1)$-connected set in G.

(b) Constructing D from G. Here, $Y = N^-(r)$.

(c) Constructing D' from D.

Figure 13.1: The graphs G, D and D' are displayed in left-to-right order, $T = \{\{s,t\},\{s,t''\},\{s',t'\}\}$ and $T' = \{\{s_1,t_1\},\{s_2,t''_1\},\{s'_1,t'_1\}\}$.

s' and t', and two new edges, $s's$ and $t't$, for each terminal pair $\{s,t\} \in T$. The modification is such that if a vertex $u \in V(D)$ belonged to ℓ terminal pairs in T, then D' would have ℓ distinct vertices corresponding to u. Now, the new set of terminal pairs is defined as $T' = \{\{s',t'\} \mid \{s,t\} \in T\}$. It is easy to see that any minimal T-r-digraph pair cut in D is also a minimal T'-r-digraph pair cut in D'. Thus, to find a superset of relevant vertices for (D,T,r,k) in the set $N^-_D(r)$, it is enough to find a superset of relevant vertices for (D',T',r,k) in the set $N^-_{D'}(r)$. Therefore, from now on we can assume that our input instance is (D',T',r,k), where the pairs in the set T' are pairwise disjoint (see Figures 13.1b and 13.1c for an illustration). Henceforth, whenever we say that a vertex is relevant (or irrelevant), we mean that it is relevant (or irrelevant) for the instance (D',T',r,k). The description of $\mathcal{A}$ is given in Algorithm 1.

The correctness of Algorithm 1 as exhibited in Lemma 13.2.3 is essentially based on creating a set up that allows the applicability of the anti-isolation lemma of Marx [157], which appeared at [178], to find and mark relevant vertices and relevant terminal pairs.

Lemma 13.2.3. *Algorithm 1 outputs a set R of size at most $64^{k+1}(k+1)^2$, which contains all relevant vertices in $N^-_D(r)$.*

Proof. Notice that Algorithm 1 returns a set R either in Line 2 or in Line 8 thus, by Lemma 13.2.2, the size of the returned set is at most $|Z| \cdot 4^k k + |Z| \leq 64^{k+1}(k+1)^2$. We now prove the correctness of the algorithm using induction on $|T'|$. When

Algorithm 1: Input is (G', T', r, k), where T' is pairwise disjoint

1 **if** $|T'| = 0$ **then**
2 **return** $\emptyset$
3 $\widehat{T} := \{s', t' \mid \{s', t'\} \in T'\}$.
4 Compute a minimum $\widehat{T}$-r-separator Z.
5 **if** $|Z| \leq 16^k \cdot 64(k+1)$ **then**
6 For each $z \in Z$, compute all important z-r-separators of size at most k.
7 Mark all the vertices in $N^-_{D'}(r)$, which are either part of the computed important separators or part of Z.
8 **return** *the set of marked vertices (call it R)*
9 **else**
10 Compute a maximum set $\mathcal{P}$ of vertex disjoint paths from $\widehat{T}$ to r (any pair of paths intersects only at r).
11 Let $X = V(\mathcal{P}) \cap \widehat{T}$. Let A be a maximum sized subset of X such that for any $\{s', t'\} \in T'$, $|A \cap \{s', t'\}| \leq 1$.
12 Let $B = \{w \mid$ there exists $w' \in A$ such that $\{w, w'\} \in T'\}$. That is, B is the set of vertices that are paired with vertices of A in the set of pairs T'.
13 Compute all important r-B-separators of size at most $2k+2$ in $\overleftarrow{D'}$.
14 Mark all vertices from B which are part of the computed important separators.
15 Let $q \in B$ be an unmarked vertex and let $\{q, q'\} \in T'$.
16 $T' := T' \setminus \{\{q, q'\}\}$ and repeat from Step 2.

$|T'| = 0$, then no vertex in $N^-_D(r)$ is relevant and the algorithm returns the correct output. Now consider the induction step where $|T'| > 0$. We have two cases based on the size of the separator Z computed in Line 4.

Case 1: $|Z| \leq 16^k \cdot 64(k+1)$**.** In this case, Lines 6-8 will be executed and Algorithm 1 will output a set R. We prove that R contains all relevant vertices in $N^-_{D'}(r)$. Towards this, we show that if S is a minimal T'-r-digraph pair cut of size at most k and $v \in N^-_{D'}(r) \cap S$, then v belongs to R. Let $S' = S \setminus \{v\}$. Since S is a minimal T'-r-digraph pair cut, S' is not a T'-r-digraph pair cut. Since S is a T'-r-digraph pair cut and S' is not a T'-r-digraph pair cut, there is a vertex $t \in \widehat{T}$ such that (i) v is reachable from t in $D' - S'$, and (ii) r is not reachable from t in $D' - S$. If $v \in Z$, then v is marked and belongs to R. Therefore, if $v \in Z$, we are done. Thus, from now on, assume that $v \notin Z$.

Claim 13.2.1. *There is a vertex $z \in Z$ that belongs to $R_{D'}(t, S)$.*

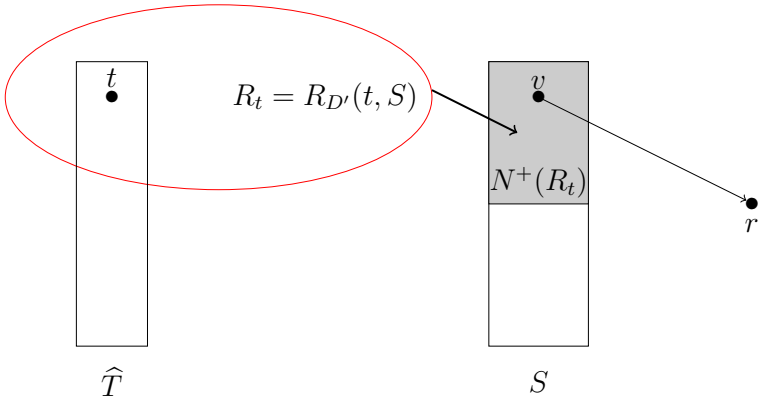


Figure 13.2: Here, the ellipse contains the set of vertices reachable from t in $D' - S$, denoted by R_t. The rectangle colored grey represents $N^+(R_t)$ which includes v

Proof. From (i), we have that $v \in R_{D'}(t, S')$. Since Z is a minimum $\widehat{T}$-r-separator, $t \in \widehat{T}$, and $v \in R_{D'}(t, S')$, we have that all paths from t to v passes through some vertex in Z. Also, since $v \in N^-_{D'}(r)$ and $v \in R_{D'}(t, S')$ and $v \notin Z$, there is a vertex $z \in Z$ that belongs to $R_{D'}(t, S)$. □

Let $R_t = R_{D'}(t, S)$ and $C = N^+_{D'}(R_t)$. Observe that $C \subseteq S$, $v \in C$ and v is reachable from z in $D' - (C \setminus \{v\})$. We claim that C is a z-r-separator. If C is not a z-r-separator, then there is a path from z to r in $D' - S$. Also, since $z \in R_{D'}(t, S)$, there is a path from t to z in $D' - S$. This implies that there is a path from t to r in $D' - S$ which is a contradiction to the statement (ii). Since v is reachable from z in $D' - (C \setminus \{v\})$, there is a minimal z-r-separator that contains v and is fully contained in C. Let $C' \subseteq C$ be a minimal z-r-separator that contains v. Since $v \in N^-_{D'}(r)$ and C' is a minimal z-r-separator, either C' is an important z-r-separator or there is an important z-r-separator of size at most k containing v which dominates C'. In either case, v is marked in Line 8 and hence, it will be in the set R (see Figure 13.2 for an illustration).

Case 2: $|Z| > 16^k \cdot 64(k+1)$**.** In this case, we prove that there, indeed, exists an unmarked vertex $q \in B$ and the pair $\{q, q'\}$ is an irrelevant terminal pair. Notice that in Line 13, we have computed all important r-B-separators of size at most $2k + 2$ for some B. By Lemma 13.2.2, the total number of vertices in all these separators together is at most $16^k \cdot 32(k+1)$. So we should have marked at most $16^k \cdot 32(k+1)$ vertices in B. We first claim that $|B| > 16^k \cdot 32(k+1)$, which ensures the existence of an unmarked vertex in B. By the definition of A, the size of A is at least $|Z|/2 > 16^k \cdot 32(k+1)$, because there are $|Z|$ vertex disjoint paths from $\widehat{T}$

to r, only intersecting at r. By the definition of B, $|B| = |A| > 16^k \cdot 32(k+1)$. Since we proved that we have only marked at most $16^k \cdot 32(k+1)$ vertices in B, this implies that there is an unmarked vertex q in B. Let q' be the unique vertex such that $\{q, q'\} \in T'$ (such a unique vertex exists because T' is pairwise disjoint).

Now we show that $\{q, q'\}$ is an irrelevant terminal pair. Let S be a minimal $(T' \setminus \{\{q, q'\}\})$-$r$-digraph pair cut of size at most k. We need to show that S is also a T'-r-digraph pair cut. We know that there are $|Z|$ vertex disjoint paths $\mathcal{P}$ from $\widehat{T}$ to r, where the paths intersect only at r. Since Z is a minimum $\widehat{T}$-r-separator, $|\widehat{T}| \geq |Z|$. Recall the definition of A and B from the description of the algorithm. Let A_r be the set of vertices in $A \setminus \{q'\}$ such that r is reachable from each vertex in A_r in $D' - S$, that is, $A_r = \{u \in A \setminus \{q'\} \mid r \in R_{D'}(u, S)\}$. Let B_r be the set of vertices in B such that r is reachable from each vertex in B_r in the graph $D' - S$, that is, $B_r = \{u' \in B \mid r \in R_{D'}(u', S)\}$. Since there are $|A|$ vertex disjoint paths from A to r (which intersect only at r) and $|S| \leq k$, we have $|A_r| \geq |A| - (k+1)$. Since S is an $(T' \setminus \{\{q, q'\}\})$-$r$-digraph pair cut, the vertices in B which are paired with a vertex in A_r are not reachable from r in $\overleftarrow{D'} - S$. This implies that $|B_r| \leq k+1$. Let $Q = S \cup B_r \cup \{q\}$. Notice that $q \in Q$ and Q is a r-B-separator in $\overleftarrow{D'}$ of size at most $2k+2$. If q is not reachable from r in $\overleftarrow{D'} - S$, then S is, indeed, a T'-r-digraph pair cut, because S is a $(T' \setminus \{\{q, q'\}\})$-$r$-digraph pair cut. In what follows we show that it is always the case, that is, q is not reachable from r in $\overleftarrow{D'} - S$. Suppose not. Since q is reachable from r in $\overleftarrow{D'} - S$ and all the vertices in $Q \setminus S$ have no out-neighbors in $\overleftarrow{D'}$ (by construction of D'), any path from r to q in $\overleftarrow{D'} - S$ will not contain any vertex from $Q \setminus \{q\}$. This implies that there is a minimal r-B-separator $Q' \subseteq Q$ containing q. Hence, either Q' is an important r-B-separator of size at most $2k+2$ or all the important r-B-separators which dominate Q' will contain q. This is implies that q is marked, which is a contradiction.

Thus, we have shown that in this case there is an irrelevant terminal pair $\{q, q'\} \in T'$, and by Observation 13.2.1 and induction hypothesis, Algorithm 1 will output the required set. □

Lemma 13.2.4. *Algorithm 1 runs in time $\mathcal{O}(|T'| \cdot n(n^{\frac{2}{3}} + m))$.*

Proof. The number of times each step of the algorithm will get executed is at most $|T'|$. By Proposition 2.4.1, Line 4 takes time $\mathcal{O}(mn)$. By Lemma 13.2.2, the time required to enumerate important separators in Lines 6 and 13 is bounded by $\mathcal{O}(4^{2k} \cdot k \cdot (n+m))$. The time required to compute $\mathcal{P}$ in Line 13 is $\mathcal{O}(mn)$ by Proposition 2.4.1. Thus, the total running time of Algorithm 1 is $\mathcal{O}(|T'|(mn + 4^{2k} \cdot k \cdot (n+m)))$.

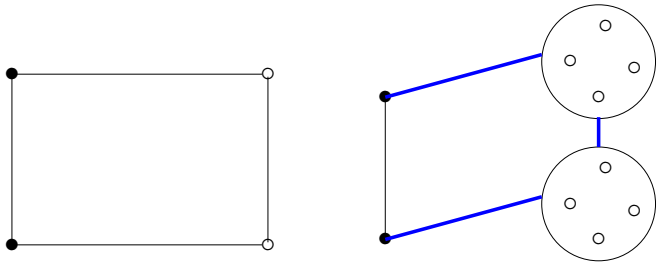


Figure 13.3: The graph at the right hand side is obtained by the reduction on the graph at the left hand side, where $k = 3$ and Y is the set of black colored vertices. Thick lines represents all possible edges between two sets of vertices.

Recall that, we could safely assume that $|V(D')| = n > 64^{k+1}(k+1)^2$. Since, $n > 64^{k+1}(k+1)^2$, $4^{2k} \cdot k < n^{\frac{2}{3}}$. Hence, the claimed running time of the algorithm follows. □

13.3 Covering Small Multicuts in a Subgraph Without Highly Connected Set

In this section, we prove that given a graph G, a set of terminal pairs $T = \{\{s_1, t_1\}, \ldots, \{s_\ell, t_\ell\}\}$ and an integer k, there is a polynomial time algorithm which finds a pair $(G^\star, T^\star)$, where $G^\star$ is an induced subgraph of G such that it has no $(k+2)$-connected sets of size $2^{\mathcal{O}(k)}$ and $T^\star \subseteq T$ such that for any $S \subseteq V(G)$ of size at most k, S is a minimal multicut of T in G if and only S is a subset of $V(G^\star)$ and S is a minimal multicut of $T^\star$ in $G^\star$. This statement is formalized in Lemma 13.3.1. Before stating Lemma 13.3.1, we give definitions of a k-connected set in a graph G and a k-connected graph.

Definition 13.3.1 (k-connected set and graphs)**.** *For any $k \in \mathbb{N}$ and a graph G, a subset Y of the vertices of G is called a* k-connected set *in G, if for any $u, v \in Y$ there are at least k internally vertex disjoint paths from u to v in G. The graph G is called a* k-connected graph *if $V(G)$ is a k-connected set in G. Equivalently, the graph G is k-connected, if the size of a mincut in G is at least k.*

Lemma 13.3.1 (Degeneracy Reduction Lemma)**.** *Let G be a graph, T be a set of terminal pairs and $k \in \mathbb{N}$. Let C be the set of all minimal multicuts of T of size at most k in G. There is a deterministic algorithm which runs in time $\mathcal{O}(|T| \cdot n^2(n^{\frac{2}{3}} + m) + n^5 \log n)$ and outputs an induced subgraph $G^\star$ of G and a subset $T^\star \subseteq T$ such that*

1. *for any $S \subseteq V(G)$ with $|S| \leq k$, S is a minimal multicut of T in G if and only if $S \subseteq V(G^\star)$ and S is a minimal multicut of $T^\star$ in $G^\star$, and*

2. *there is no $(k+2)$-connected set of size at least $64^{k+2} \cdot 4(k+2)^2$ in $G^\star$.*

The proof of Lemma 13.3.1 requires some auxiliary lemmas which we discuss below. Recall the definition of the problem Multicut from Section 15.2. Let (G, T, k) be an instance of Multicut. We say that a vertex $v \in V(G)$ is *irrelevant* to (G, T, k) if no minimal multicut of G of size at most k in G contains v. Lemma 13.3.2 states that if a graph has a sufficiently large $(k+2)$-connected set, then many of its vertices are irrelevant to the given Multicut instance. Such a statement is deduced by establishing a relation between the multicuts of the given instance and the digraph pair cuts of practically the same instance. This relationship then relates the irrelevant vertices to the instance of Multicut with the irrelevant vertices to the instance for Digraph Pair Cut.

Lemma 13.3.2. *Let G be a graph, T be a set of terminal pairs, $k \in \mathbb{N}$ and Y be a $(k+1)$-connected set in G. Let D be a digraph obtained by adding a new vertex r, whose in-neighbors are the vertices of Y, and replacing each edge of G by two arcs with the same endpoints and opposite orientations. Then, any irrelevant vertex of Y to the instance (D, T, r, k) of* Digraph Pair Cut *is also an irrelevant vertex to the instance (G, T, k) of* Multicut.

Proof. The construction of D from G is illustrated in Figures 13.1a and 13.1b. Suppose there exists $v \in Y$ which is relevant to the instance (G, T, k) of Multicut. Then, there exists a minimal multicut, say C, of T in G of size at most k such that $v \in C$. We first claim that C is an T-r-digraph pair cut in D. Suppose not. Then, there is a pair $\{s, t\} \in T$ such that there is a path from s to r and t to r in $D - C$. Since the in-neighbors of r are the vertices of Y, there exist $u_1, u_2 \in Y$, u_1 may be equal to u_2, such that there are two paths, one from s to u_1 and other from t to u_2, in $G - C$. If $u_1 = u_2$, then s and t are in the same connected component of $G - C$, which is a contradiction. Otherwise, since Y is a $(k+1)$-connected set in G and $u_1, u_2 \in Y$, there are $k+1$ internally vertex disjoint paths from u_1 to u_2. Since $|C| \leq k$, there exists a path between u_1 and u_2 in $G - C$, and hence a path between s and t in $G - C$, which is a contradiction.

We next show that there exists $C' \subseteq C$ such that $v \in C'$ and C' is a minimal T-r-digraph pair cut in D. This will prove that v is relevant to the instance (D, T, r, k) of Digraph Pair Cut thereby proving the claim. Since C is an T-r-digraph pair

cut in D, there exists $C' \subseteq C$ such that C' is a minimal T-r-digraph pair cut in D. Suppose $v \notin C'$. Since C is a minimal multicut of T in G, there exists a terminal pair $(s,t) \in T$ such that there is a path from s to t in $G - (C \setminus \{v\})$. In particular, there is a path from s to v and t to v in $G - (C \setminus \{v\})$. Since $v \in Y$, by the construction of D, v is an in-neighbor of r. Hence, there is a path from s to r and t to r in $D - (C \setminus \{v\})$. Thus, if $v \notin C'$, then C' is not an T-r-digraph pair cut in D, which is a contradiction. □

Using Lemmas 13.2.1 and 13.3.2, one can find irrelevant vertices to the given instance of MULTICUT, if the graph in the instance has a $(k+1)$-connected set Y of size strictly more than $64^{k+1}(k+1)^2$ and the set Y is explicitly given as input. So the next task is to design an algorithm that finds a $(k+1)$-connected set in a graph of a given size, if it exists.

13.4 Finding Large Connected Sets

The goal of this section is to prove Lemma 13.4.1.

Lemma 13.4.1. *There is an algorithm which given a graph G and $k, d \in \mathbb{N}$, $k \leq d$, runs in time $\mathcal{O}(n^4 \log n)$, and either concludes that there is no k-connected set of size at least $4d$ in G or outputs a k-connected set in G of size at least $d+1$.*

The proof of Lemma 13.4.1 requires an auxiliary lemma (Lemma 13.4.3) which we prove next. Lemma 13.4.3 is an algorithmic version of the following famous result of Mader [152] which says that if a graph has large average degree (or degeneracy), then it contains a $(d+1)$-connected subgraph.

Lemma 13.4.2 ([152])**.** *Let $d \in \mathbb{N} \setminus \{0\}$. Every graph G with average degree at least $4d$ has a $(d+1)$-connected subgraph.*

The proof of Lemma 13.4.2 given in [76, 199] can be modified to get a polynomial time algorithm. The following lemma, an algorithmic version of Lemma 13.4.2, is written in terms of the degeneracy of the graph.

Lemma 13.4.3. *There is an algorithm which, for any $d \in \mathbb{N} \setminus \{0\}$, given a graph G with degeneracy at least $4d$, runs in time $\mathcal{O}(n^2 m \log n)$ and outputs a $(d+1)$-connected subgraph of G.*

Proof. The algorithm first constructs a subgraph H of G which has minimum degree at least $4d$. To do so, first set $H := G$. If the minimum degree of H is at least $4d$, then we are done. Otherwise, let v be a vertex of H of degree at most $4d-1$. Set $H := H - v$ and repeat this process. Since the degeneracy of G is at least $4d$, the procedure will end up in a subgraph of G that has minimum degree at least $4d$. The naive implementation of the above procedure takes time $\mathcal{O}(mn)$.

Claim 13.4.1. *For any $d \in \mathbb{N} - \{0\}$, if the minimum degree of a graph H is at least $4d$, then $|V(H)| \geq 2d+1$ and $|E(H)| \geq 2d(|V(H)| - d - \frac{1}{2})$.*

Proof. Since minimum degree of H is at least $4d$, clearly $|V(H)| \geq 4d+1 \geq 2d+1$. Also, since $\sum_{v \in V(H)} deg_G(v) = 2|E(H)|$ and for all $v \in V(H)$ $deg_G(v) \geq 4d$, $|E(H)| \geq 2d|V(H)| \geq 2d(|V(H)| - d - \frac{1}{2})$. □

From Claim 13.4.1, we conclude that $|V(H)| \geq 2d+1$ and $|E(H)| \geq 2d(|V(H)| - d - \frac{1}{2})$. Thus, from the following claim (Claim 13.4.2), one can infer that H has a $(d+1)$-connected subgraph. Using this claim, we will later give an algorithm that actually computes a $(d+1)$-connected subgraph of H, whose correctness will follow from the proof of Claim 13.4.2.

Claim 13.4.2. *Let H be any graph and $d \in \mathbb{N} \setminus \{0\}$ such that $|V(H)| \geq 2d+1$ and $|E(H)| \geq 2d(|V(H)| - d - \frac{1}{2})$. Then H has a $(d+1)$-connected subgraph.*

Proof. We prove the claim using induction on $|V(H)|$. The base case of the induction is when $|V(H)| = 2d+1$. From the premises of the claim, if $|V(H)| = 2d+1$, $|E(H)| \geq 2d(2d+1-d-\frac{1}{2}) = 2d(d+\frac{1}{2}) = \binom{2d+1}{2}$. Since a graph on $2d+1$ vertices can have at most $\binom{2d+1}{2}$ edges, H is a clique on $2d+1$ vertices, which is a $(d+1)$-connected graph. Now consider the induction step where $|V(H)| > 2d+1$. Suppose there is a vertex $v \in V(H)$ such that $deg_H(v) \leq 2d$. Then $|V(H-v)| \geq 2d+1$ and $|E(H-v)| \geq |E(H)| - 2d \geq 2d(|V(H-v)| - d - \frac{1}{2})$. Thus, from the induction hypothesis, there is a $(d+1)$-connected subgraph in $H - v$. From now on, we can assume that the degree of each vertex in H is at least $2d+1$. Suppose H itself is a $(d+1)$-connected graph, then we are done. If not, then there exists a mincut, say Z, of H, of size at most d. Let $U_1 \uplus U_2$ be a partition of $V(G) \setminus Z$ such that there is no edge between a vertex in U_1 and a vertex in U_2, and $U_1, U_2 \neq \emptyset$. Let $A = Z \cup U_1$ and $B = Z \cup U_2$. We claim that either $H[A]$ or $H[B]$ satisfy the premises of the claim. Notice that all the neighbors of any vertex $s \in U_1$ are in A and all the neighbors of any vertex $t \in U_2$ are in B. Also since, $deg_H(s), deg_H(t) \geq 2d+1$, we have that $|A| \geq 2d+1$ and $|B| \geq 2d+1$. Thus,the vertex set cardinality constraint stated in the premise of the claim is met for both $H[A]$ and $H[B]$. Suppose that, the edge set cardinality constraint stated in the premise of the claim is not met for both $H[A]$

and $H[B]$. Then we have the following.

$$\begin{aligned} |E(H)| &\leq |E(H[A])| + |E(H[B])| \\ &< 2d(|A| - d - \frac{1}{2}) + 2d(|B| - d - \frac{1}{2}) \\ &= 2d(|A| + |B| - 2d - 1) \\ &\leq 2d(|V(H)| + d - 2d - 1) \\ &< 2d(|V(H)| - d - \frac{1}{2}). \end{aligned}$$

This is a contradiction to the fact that $|E(H)| \geq 2d(|V(H)|-d-\frac{1}{2})$. Therefore, either $H[A]$ or $H[B]$ satisfy the premises of the claim. Moreover, notice that $|A| < |V(H)|$ and $|B| < |V(H)|$, because $U_1, U_2 \neq \emptyset$. Thus, by the induction hypothesis the claim follows. □

The above proof can easily be turned into an algorithm. This is explained below. Our algorithm for finding a $(d+1)$-connected subgraph of H works as follows. It first tests whether H itself is a $(d+1)$-connected graph - this can be done by computing a mincut of H (using the algorithm of Proposition 2.4.2) and then testing whether the size of a mincut of H is at least $d+1$. If H is a $(d+1)$-connected graph, then our algorithm outputs $V(H)$. Otherwise, if there is a vertex of degree at most $2d$ in H, then it recursively finds a $(d+1)$-connected subgraph in $H - v$. If all the vertices in H have degree at least $2d+1$, then it finds a mincut Z in H (using the algorithm of Proposition 2.4.2). It then constructs vertex sets A and B as mentioned in the proof of Claim 13.4.2. It is proved in Claim 13.4.2 that either $H[A]$ or $H[B]$ satisfy the premises of Claim 13.4.2, and it can be tested in linear time whether a graph satisfies the premises of Claim 13.4.2. If $H[A]$ satisfies the premises of Claim 13.4.2, then our algorithm recursively finds a $(d+1)$-connected subgraph in $H[A]$. Otherwise, our algorithm recursively find a $(d+1)$-connected subgraph in $H[B]$.

Note that this algorithm makes at most n recursive calls and in each recursive call it runs the algorithm of Proposition 2.4.2 and does some linear time testing. Thus, given a graph H of minimum degree at least $2d$, this algorithm runs in time $\mathcal{O}(mn^2 \log n)$ and outputs a $(d+1)$-connected subgraph of H. The algorithm claimed in the lemma first constructs a subgraph H of G of minimum degree at least $2d$, as described earlier, in time $\mathcal{O}(mn)$ and takes additional $\mathcal{O}(n^2 m \log n)$ time to output a $(d+1)$-connected subgraph of H. Thus, the total running time of this algorithm is $\mathcal{O}(n^2 m \log n)$. □

We are now ready to give the proof of Lemma 13.4.1.

Proof of Lemma 13.4.1. The algorithm first constructs an auxiliary graph $G^\star$ as follows. The vertex set of $G^\star$ is $V(G)$ and for any $u, v \in V(G^\star)$,$uv \in E(G^\star)$ if and only if the size of a minimum u-v-separator in G is at least k (that is, there are at least k internally vertex disjoint paths from u to v in G). It then checks whether the degeneracy of $G^\star$ is at least $4d-1$ or not. If the degeneracy of $G^\star$ is strictly smaller than $4d-1$, then the algorithm outputs that there is no k-connected set in G of size at least $4d$. Otherwise, the degeneracy of $G^\star$ is at least $4d-1 \geq 4(d-1)$. In this case, the algorithm applies the algorithm of Lemma 13.4.3 for $(G^\star, d-1)$, which outputs a d-connected subgraph H of $G^\star$. Since H is a d-connected subgraph, $|V(H)| \geq d+1$. Since, $k \leq d$, H is k-connected in $G^\star$. The algorithm outputs $V(H)$ as the k-connected set in G.

To prove the correctness of the algorithm, we need to prove the following two statements.

1. When our algorithm reports that there is no k-connected set in G of size at least $4d$, that is, when degeneracy of $G^\star$ is at most $4d-2$, then the graph G has no k-connected set of size at least $4d$.
2. When our algorithm outputs a set, that is, when degeneracy of $G^\star$ is at least $4d-1$, then the set outputted is a k-connected set in G of size at least $d+1$. In other words, if degeneracy of $G^\star$ is at least $4d-1$, then the set $V(H)$ is k-connected in G and has size at least $d+1$.

For the proof of the first statement, observe that when G has a k-connected set, say Y, of size at least $4d$, then $G^\star[Y]$ is a clique. Hence, the degeneracy of $G^\star$ is at least $4d-1$. For the proof of the second second, we have already argued that the size of $V(H)$ is at least $d+1$ and that $V(H)$ is a k-connected set in $G^\star$. We will now prove $V(H)$ is a k-connected set in G.

Claim 13.4.3. *$V(H)$ is a k-connected set in G.*

Proof. Observe that, it is enough to show that for any $u, v \in V(H)$ and any $C \subseteq V(G) \setminus \{u, v\}$ of size strictly smaller than k, there is a path from u to v in $G - C$. Since H is a k-connected subgraph of $G^\star$, there is a path from u to v in $G^\star - C$. Let $w_1 w_2 \ldots w_\ell$, where $w_1 = u$ and $v = w_\ell$, be a path from w_1 to w_ℓ in $G^\star - C$. Since for any $i \in [\ell - 1]$, $w_i w_{i+1} \in E(G^\star)$, there are at least k vertex disjoint paths from w_i to w_{i+1} in G. Also, since $|C| < k$, there is a path from w_i to w_{i+1} in $G - C$. This implies that there is a path from $w_1 = u$ to $w_\ell = v$ in $G - C$, proving that H is a k-connected set in G. □

This finishes the proof of correctness of our algorithm. We now analyze the total running time of the algorithm. The graph $G^\star$ can be constructed in time $\mathcal{O}(k \cdot n^2(n+m))$ using Proposition 2.4.1. Also, checking whether the graph has degeneracy at least $4d-1$ can be done in time $\mathcal{O}(mn)$. Since $G^\star$ could potentially have $\mathcal{O}(n^2)$ edges, by Lemma 13.4.3 the subgraph H can be computed in time $\mathcal{O}(n^4 \log n)$. Thus the total running time of our algorithm is $\mathcal{O}(n^4 \log n)$. □

Lemma 13.4.4. *There is an algorithm that given a graph G, a set of terminal pairs T and $k \in \mathbb{N}$, runs in time $\mathcal{O}(|T| \cdot n(n^{\frac{2}{3}}+m)+n^4 \log n)$ and, either correctly concludes that G does not contain a $(k+1)$-connected set of size at least $64^{k+1} \cdot 4(k+1)^2$ or finds an irrelevant vertex for the instance (G,T,k) of* Multicut.

Proof. Let $d = 64^{k+1}(k+1)^2$. Our algorithm first runs the algorithm of Lemma 13.4.1 on the instance $(G, k+1, d)$. If this algorithm (of Lemma 13.4.1) concludes that there is no $(k+1)$-connected set in G of size at least $4d$, then our algorithm returns the same. Otherwise, the algorithm of Lemma 13.4.1 outputs a $(k+1)$-connected set Y in G of size at least $d+1$. Our algorithm then creates a digraph D as mentioned in Lemma 13.3.2. It then applies the algorithm of Lemma 13.2.1 and computes a set Z of irrelevant vertices for the instance (D,T,r,k) of Digraph Pair Cut in the set Y. From Lemma 13.3.2, Z is also a set of irrelevant vertices for the instance (G,T,k) of Multicut. Since $|Y| \geq d+1$ and the number of relevant vertices for (D,T,r,k) in the set Y is at most d (from Lemma 13.2.1), $Z \neq \emptyset$. Our algorithm then outputs an arbitrary vertex v from the set Z as an irrelevant vertex for (G,T,k).

By Lemmas 13.2.1 and 13.4.1, the total running time of our algorithm is $\mathcal{O}(|T| \cdot n(n^{\frac{2}{3}}+m)+n^4 \log n)$. □

Lemma 13.4.5. *There is an algorithm which given as input a graph G, a set of terminal pairs T and $k \in \mathbb{N}$, runs in time $\mathcal{O}(|T| \cdot n(n^{\frac{2}{3}}+m)+n^4 \log n)$ and either concludes that there is no $(k+2)$-connected set of size at least $64^{k+2} \cdot 4(k+2)^2$ in G, or outputs a vertex $v \in V(G)$ such that for any $S \subseteq V(G)$ with $|S| \leq k$, S is a minimal multicut of T in G if and only if $S \subseteq V(G) \setminus \{v\}$ and S is a minimal multicut of $T' = \{\{s,t\} \in T \mid v \notin \{s,t\}\}$ in $G - v$.*

Proof. This algorithm runs the algorithm of Lemma 13.4.4 on the instance $(G,T,k+1)$. If the algorithm of Lemma 13.4.4 outputs that there is no $(k+2)$-connected set of size $64^{k+2} \cdot 4(k+2)^2$ in G, then our algorithm reports the same. Otherwise, let v be the vertex returned by the algorithm of Lemma 13.4.4, which is irrelevant for $(G,T,k+1)$ (from Lemma 13.4.4), then it also returns v. The running time of our algorithm follows from Lemma 13.4.4.

We now prove the correctness of this algorithm. For the forward direction, let $S \subseteq V(G)$ be such that $|S| \leq k$ and S is a minimal multicut of T in G. By the definition of an irrelevant vertex for the instance $(G, T, k+1)$, we conclude that $S \subseteq V(G) \setminus \{v\}$. Since $T' \subseteq T$ and G is a supergraph of $G - v$, S is a multicut of T' in $G - v$. Suppose, for the sake of contradiction, that S is not a minimal multicut of T' in $G - v$. Then, there exists $S' \subsetneq S$ such that S' is a minimal multicut of T' in $G - v$. If S' is multicut of T in G, then we contradict the fact that S is a minimal multicut of T in G. Otherwise, there exists $S'' \subseteq S' \cup \{v\}$ and $v \in S''$, such that S'' is a minimal multicut of T in G, which contradicts that v is an irrelevant vertex for $(G, T, k+1)$. Hence we have proved that S is a minimal multicut of T' in $G - v$.

For the backward direction, let $S \subseteq V(G) \setminus \{v\}$ be such that S is a minimal multicut of T' in $G - v$. If S is a multicut of T in G, then S has to be a minimal multicut of T in G, as otherwise it would contradict that S is a minimal multicut of T' in $G - v$. Otherwise, $S \cup \{v\}$ is a multicut of T in G, because all the terminal pairs in $T \setminus T'$ contain v. Let $S' \subseteq S \cup \{v\}$ be a minimal multicut of T in G. Note that $v \in S'$ and $|S'| \leq k+1$. This contradicts the fact that v is an irrelevant vertex for $(G, T, k+1)$. □

Lemma 13.3.1 can easily be proved by applying Lemma 13.4.5 at most n times.

Chapter 14

Faster Algorithms and Combinatorial Bounds for Independent Feedback Vertex Set

In this chapter, we design two fast algorithms for Independent Feedback Vertex Set (IFVS) - an FPT algorithm and a moderately exponential time algorithm. We also give a combinatorial bound on the number of minimal independent feedback vertex sets in a graph. We begin by formally defining the problem.

Independent Feedback Vertex Set (IFVS) **Parameter:** k
Input: An undirected graph G on n vertices and a positive integer k.
Question: Is there a set S of vertices of size at most k such that $G - S$ has no cycles and S is an independent set in G?

Such a set S is called an independent feedback vertex set of G.

IFVS and Parameterized Complexity. FVS together with Vertex Cover is one of the most well studied problem in the field of parameterized complexity [39, 48, 132, 182]. The other variants of FVS on undirected graphs that have been studied extensively, include, Subset FVS [72, 148, 121], Group FVS [71, 113, 121], Connected FVS [162], Simultaneous FVS [3] and indeed IFVS [162, 196, 200]. The current champion algorithms for FVS are: a randomized algorithm with running time $\mathcal{O}^\star(3^k)$ [70] and a deterministic algorithm running in time $\mathcal{O}^\star(3.619^k)$ [132]. Misra et al. [162] introduced IFVS in 2011 (in the conference version of the cited paper) as a generlization of FVS and gave an algorithm with running time $\mathcal{O}^\star(5^k)$.

They also designed a polynomial kernel of size $\mathcal{O}(k^3)$ for the problem. Later, Song claimed a deterministic algorithm with running time $\mathcal{O}^\star(4^k)$ for the problem [196]. However, the algorithm of Song [196] does not seem to be correct. We give a sketch of the fundamental problem in the arguments used and a family of counter-examples to his algorithm in the Section 14.2.2. Tamura et al. [200] studied IFVS on special graph classes and showed that the problem remains NP-complete even on planar bipartite graphs of maximum degree four. They also designed linear time algorithms for graphs of bounded treewidth, chordal graphs and cographs. Finally, they gave an algorithm with running time $\mathcal{O}(2^{\mathcal{O}(\sqrt{k}\log k)}n)$ for IFVS on planar graphs. Our first main result is the following result regarding IFVS.

Theorem 14.0.1. *There is an algorithm for* IFVS *running in time* $\mathcal{O}^\star(4.1481^k)$.

We would like to point out that more recently, our result in Theorem 14.0.1 has been improved by Li and Pilipczuk [140], who designed an algorithm for IFVS with running time $\mathcal{O}^\star((1+\phi^2)^k)$, where ϕ is the golden ratio, $\phi < 1.619$. This running time matches the running time of the best known algorithm for FVS.

Our new algorithm is based on iterative compression and the subroutine for iterative compression is based on branching. The branching algorithm itself exploits (a) the fact that once we select a vertex in the independent feedback vertex set then all its neighbors must be in the forest; and (b) an interesting variation of measure used for analyzing the fastest known deterministic algorithm for FVS [132]. Finally, we also observe that the randomized algorithm designed for FVS, running in time $\mathcal{O}^\star(3^k)$ [70], can be adapted to design a randomized $\mathcal{O}^\star(3^k)$ time algorithm for IFVS.

IFVS and Exact Exponential Time Algorithms. In exact exponential time algorithms, the objective is to design an algorithm for optimization version of a problem that is better than the naïve brute force algorithm. In particular, for FVS the goal will be to design an algorithm that runs in time c^n, $c < 2$ a constant, and finds a minimum sized set S such that $G - S$ is a forest. We refer to the book of Fomin and Kratsch for more details regarding moderately exponential time algorithms [98]. Obtaining a non-trivial exact algorithm for FVS was open for quite some time before Razgon obtained an algorithm with running time $\mathcal{O}(1.8899^n)$ [185]. Later this algorithm was improved to $\mathcal{O}^\star(1.7347^n)$ [103]. Recently, Fomin et al. [94] obtained an interesting result relating parameterized algorithms and exact algorithms. Roughly speaking, they showed that if a problem (satisfying some constraints) has $\mathcal{O}^\star(c^k)$ time algorithm parameterized by the solution size, then there is an exact algorithm running in time $\mathcal{O}^\star(\left(2-\frac{1}{c}\right)^n)$. Both FVS and IFVS satisfies the required con-

straints and thus we immediately obtain the following exact algorithm for IFVS: (a) a randomized algorithm running in time $\mathcal{O}^\star(\left(2-\frac{1}{3}\right)^n) = \mathcal{O}^\star(1.6667^n)$; and (b) a deterministic algorithm running in time $\mathcal{O}^\star(\left(2-\frac{1}{4.1481}\right)^n) = \mathcal{O}^\star(1.7590^n)$. We give a recursive algorithm based on classical measure and conquer [96, 98] and design faster algorithm than the both mentioned algorithms. In particular, we prove the following theorem.

Theorem 14.0.2. *There is an algorithm for* IFVS *running in time* $\mathcal{O}^\star(1.5981^n)$.

Combinatorial Upper Bounds. In our final section we address the following question: How many minimal ifvses are there in any graph on n vertices? Proving an upper bound on the number of combinatorial structures is an old and vibrant area. Some important results in this area include an upper bound of

- $3^{n/3}$ on the number of maximal independent sets in a graph [167].
- 1.667^n on the number of minimal feedback vertex sets in a tournament [94].
- 1.8638^n on the number of minimal feedback vertex sets in a graph [95, 97].

One can easily observe that every minimal ifvs is also a minimal feedback vertex set. Thus, an upper bound of 1.8638^n on the number of minimal ifvses in any graph on n vertices follows by [95]. As our final result, we give an improved upper bound on the number of minimal ifvses in any graph on n vertices.

Theorem 14.0.3. *A graph G on n vertices has at most* 1.7480^n *minimal ifvses.*

Let n be divisible be 3 and G be a graph that is union of $n/3$ vertex disjoint triangles. Then any minimal ifvs must contain exactly one vertex from each of $n/3$ triangles and thus G has $3^{n/3}$ number of minimal ifvses. Closing the gap between $3^{n/3}$ and 1.7480^n remains an interesting open problem.

14.1 Some Preliminaries

In this chapter, we consider finite graphs possibly having loops and multi-edges. We also use the convention that a loop at a vertex v contributes two to its degree. We note that both a double edge and a loop are cycles. A tree T rooted at $r \in V(T)$ is called as a *star* if $E(T) = \{(v, r) \mid v \in V(T) \setminus \{r\}\}$.

Let $W \subseteq V(G)$ and $H = G - W$. We define certain useful vertices in $V(H)$. We call a vertex $v \in V(H)$, a *nice vertex* if $d_H(v) = 0$ and $d_G(v) = 2$ i.e., both the neighbours of v belong to the set W. Similarly, we call a vertex $v \in V(H)$, a *tent* if $d_H(v) = 0$ and $d_G(v) = 3$. A *feedback vertex set* is a subset $S \subseteq V(G)$ such that $G - S$ is a forest.

14.2 FPT Algorithm for Independent Feedback Vertex Set

In this section we give an FPT algorithm for IFVS running in time $\mathcal{O}^\star(4.1481^k)$. Given an input (G, k), the algorithm starts by computing a feedback vertex set Z in G. A feedback vertex set in G of size at most k (if it exists) can be computed in time $\mathcal{O}^\star(3.619^k)$ using the algorithm given in [133]. If there is no feedback vertex set of size at most k, then we conclude that (G, k) is a No instance of ifvs since an ifvs is also a feedback vertex set in G.

We let $H = G - Z$. The algorithm either outputs an ifvs in G of size at most k or correctly conclude that (G, k) is a No instance of IFVS. The algorithm guesses a subset $Z' \subseteq Z$, such that for an ifvs X in G, $X \cap Z = Z'$. For each of the guess Z', the algorithm does the following. If $G[Z']$ is not an independent set then it concludes that there is no ifvs X in G such that $Z' \subseteq X$. Otherwise, $G[Z']$ is an independent set. Let $W = Z \setminus Z'$. If $G[W]$ is not a forest, then their is no ifvs X such that, $X \cap Z = Z'$. Therefore, the guess Z' is not correct and the algorithm rejects this guess. Otherwise, it deletes the vertices in Z' and tries to find an ifvs $S \subseteq V(H) \setminus W$ of size at most $k - |Z'|$. Note that any vertex $v \in N_H(Z')$ cannot be part of the solution. Therefore, the algorithm adds the vertices in $N_H(Z')$ to a set $\mathcal{R}$. The set $\mathcal{R}$ consists of those vertices which cannot be included in ifvs in order to maintain the independence of the vertices included in the solution. The algorithm calls the sub-routine Disjoint Independent Feedback Vertex Set (Dis-IFVS) on the instance $(G, W, \mathcal{R}, k - |Z'|)$ to find an ifvs $X \subseteq V(H) \setminus (W \cup \mathcal{R})$. In Section 14.2.1 we give an algorithm for Dis-IFVS, which given an instance $(G, W, \mathcal{R}, k)$ either finds an ifvs $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ of size at most k or correctly concludes that there does not exits such an ifvs. Moreover, the algorithm for Dis-IFVS runs in time $\mathcal{O}^\star(3.1481^k)$.

Theorem 1. *There is an algorithm for* IFVS *running in time* $\mathcal{O}^\star(4.1481^k)$.

Proof. Given an instance (G, k) of IFVS, the algorithm computes a feedback vertex set Z in G of size at most k (if it exists) in time $\mathcal{O}^\star(3.619^k)$. If there is no feedback vertex set of size at most k, it correctly concludes that (G, k) is a No instance. Otherwise, for each $Z' \subseteq Z$, either it correctly concludes that Z' is a wrong guess (for extending it to an ifvs) or runs the algorithm for Dis-IFVS on the instance $(G, W, \mathcal{R}, k-|Z'|)$. Here, the instance $(G, W, \mathcal{R}, k-|Z'|)$ is created as described above in the description of the algorithm. The correctness of the algorithm follows from the correctness of the algorithm for Dis-IFVS and the fact that all possible intersections of the solution with Z are considered. The running time of the algorithm is given by the following equation: $3.619^k \cdot n^{\mathcal{O}(1)} + \sum_{i=0}^{k} \binom{k}{i} \cdot 3.1481^{k-i} \cdot n^{\mathcal{O}(1)} \leq 4.1481^k \cdot n^{\mathcal{O}(1)}$. This concludes the proof. □

14.2.1 Algorithm for Disjoint Independent Feedback Vertex Set.

We give an algorithm for Disjoint Independent Feedback Vertex Set running in time $\mathcal{O}^\star(3.1481^k)$.

Disjoint Independent Feedback Vertex Set (Dis-IFVS) **Parameter:** k
Input: An undirected (multi) graph G, a fvs W in G, $\mathcal{R} \subseteq V(G) \setminus W$ and, an integer k.
Question: Does G has an ifvs $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ such that $\lvert S\rvert \leq k$?

The algorithm for Dis-IFVS either applies some reduction rules or branches on a vertex in $V(G) \setminus W$. The algorithm branches on a vertex in $V(G) \setminus W$ only when (a) none of the reduction rules are applicable; and (b) we are not in the case where we can solve the problem in polynomial time. Let $H = G - W$. We arbitrary root the trees in H at some vertex (preferably a vertex v with $d_H(v) > 2$). We will be using the following measure μ associated with the instance $(G, W, \mathcal{R}, k)$ to bound the number of nodes of the search tree.

$$\mu = \mu(G, W, \mathcal{R}, k) = 2k + \rho(W) - (\eta + 2\tau)$$

Here, $\rho(W)$ is the number of connected components in W, η denotes the number of nice vertices in $V(H) \setminus \mathcal{R}$ and τ denotes the number of tents in $V(H) \setminus \mathcal{R}$. See preliminaries for the definitions of nice vertices and tents.

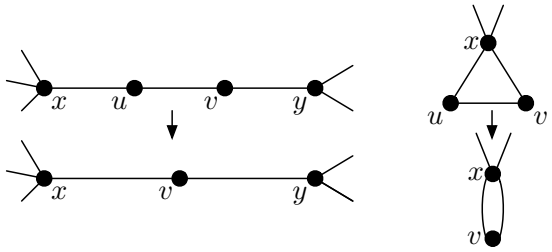


Figure 14.1: Reduction Rule 2

Now we describe all the reduction rules that will be used by the algorithm. The first two reduction rules get rid of vertices of degree at most one and consecutive vertices of degree two in the graph. The safeness of these reduction rules follow from [162]. The safeness of first two reduction rules follow from [162].

Reduction Rule 14.2.1. *Delete vertices of degree at most one since they do not participate in any cycle.*

Reduction Rule 14.2.2. *Let u, v be two adjacent degree two vertices in the input graph G, and x, y be the other neighbors of u, v respectively. If $u \in \mathcal{R}$, then delete u and add the edge (x, v). Otherwise, delete v and add the edge (x, v) (see Figure 14.1).*

When applying Reduction Rule 14.2.2, if both the degree two vertices belong to $\mathcal{R}$, then the choice of deleting one of them and adding an edge between its neighbors is arbitrary. Observe that the measure μ does not increase after the application of Reduction Rules 14.2.1 or 14.2.2.

Reduction Rule 14.2.3. *If $k < 0$, then return that $(G, W, \mathcal{R}, k)$ is a* NO *instance.*

Reduction Rule 14.2.4. *If there is a vertex $v \in \mathcal{R}$ such that v has 2 neighbors in the same component of W. Then, return that $(G, W, \mathcal{R}, k)$ is a* NO*instance.*

Reduction Rule 14.2.5. *If there is a vertex $v \in \mathcal{R}$ such that v has a neighbor in W, then remove v from $\mathcal{R}$ and add v to W. That is, we solve the instance $(G, W \cup \{v\}, \mathcal{R} \setminus \{v\}, k)$.*

Observe that by moving v to W we do not increase the number of connected components of $G[W \cup \{v\}]$.

Reduction Rule 14.2.6. *If there is a vertex $v \in V(H) \setminus \mathcal{R}$ such that v has at least 2 neighbors in the same connected component of W, then remove v from G and add the vertices in $N_H(v)$ to $\mathcal{R}$. That is, the resulting instance is $(G - \{v\}, W, \mathcal{R} \cup N_H(v), k - 1)$.*

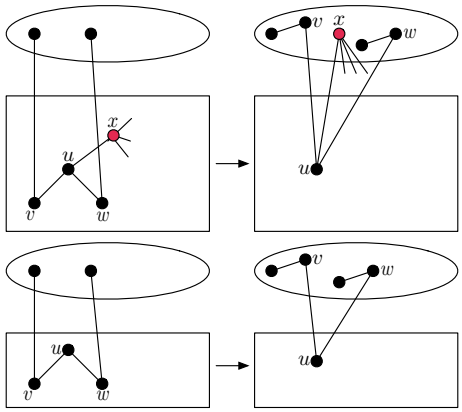


Figure 14.2: Illustration of Reduction Rule 14.2.8.

In this case it is easy to observe that v must belong to any ifvs.

Reduction Rule 14.2.7. *If there is a vertex $u \in \mathcal{R}$ such that there is a leaf v in H adjacent to u and $d_W(v) \leq 2$. Then remove u from $\mathcal{R}$ and include u in W i.e., the resulting instance is $(G, W \cup \{u\}, \mathcal{R} \setminus \{u\}, k)$.*

Reduction Rule 14.2.8. *Let T be a (rooted) tree in H and $u \in V(T) \setminus \mathcal{R}$. Furthermore, let T_u be the subtree in T rooted at u, and V_u be the set of leaves (not including u) in T_u. Suppose that T_u satisfies the following conditions:*

1. *T_u is a star such that $1 \leq |V_u| \leq 2$;*
2. *For each $x \in V_u$, we have $|N_W(x)| = 1$;*
3. *Either $T = T_u$ or, the (unique) parent x of u in T belongs to the set $\mathcal{R}$.*

Let $X = \{x\}$ if x exists, otherwise, $X = \emptyset$. Then, add the vertices in $V_u \cup X$ to W and remove X from $\mathcal{R}$ i.e., the resulting instance is $(G, W \cup V_u \cup X, \mathcal{R} \setminus X, k)$.

Lemma 14.2.1. *Reduction Rule 14.2.4 is safe.*

Proof. Let x, y be two neighbors of $v \in \mathcal{R}$ that are present in the same component of W. Since x, y belong to the same component of W, there is a path P in W from x to y. But then, $G[V(P) \cup \{v\}]$ contains a cycle, with v being the only vertex not in W. Therefore, there cannot exist an ifvs, $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ in G. This concludes the proof. □

Lemma 14.2.2. *Reduction Rule 14.2.5 is safe. Furthermore, the measure μ does not increase after application of Reduction Rule 14.2.5.*

Proof. Let $v \in \mathcal{R}$ be a vertex such that v has a neighbor in W. Note that any ifvs, $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ in G does not contain v. Moreover, since v has a neighbor in W, adding v to W does not increase the number of connected components in W. This implies that μ does not increase. □

Lemma 14.2.3. *Reduction Rule 14.2.6 is safe. Furthermore, the measure μ does not increase after application of Reduction Rule 14.2.6.*

Proof. Let $v \in V(H) \setminus \mathcal{R}$ be a vertex such that v has 2 neighbors, say x, y, in the same connected component of W. Since x, y belong to the same component of W, there is a path P in W from x to y. But then, $G[V(P) \cup \{v\}]$ contains a cycle, with v being the only vertex not in W. Therefore, any ifvs $S \subseteq W$ must include v and hence avoid $N_H(v)$.

When we delete v from G and decrease k by 1, the number of connected components in W remains the same. If v was either a nice vertex or a tent then $\eta + 2\tau$ can decrease at most by 2. Therefore, the measure μ in the resulting instance cannot increase. This concludes the proof. □

Lemma 14.2.4. *Reduction Rule 14.2.7 is safe and the measure μ does not increase after its application.*

Proof. Let $u \in \mathcal{R}$ be a vertex such that there is a leaf v in H adjacent to u such that $d_W(v) \leq 2$. Observe that no solution to Dis-IFVS can contain u. Therefore, we only need to show that the measure μ does not increase. When we add u to W, the number of components in W can increase by 1. But then v becomes either a nice vertex or a tent. Therefore, $\eta + 2\tau$ decreases at least by 1. This together with the fact that k remains the same imply that μ cannot increase. □

Lemma 14.2.5. *Reduction Rule 14.2.8 is safe and the measure μ does not increase after its application.*

Proof. Let T be a tree in H and $u \in V(T) \cap (V(H) \setminus \mathcal{R})$ such that the tree, T_u, rooted at u is a star. That is, all the children of u are leaves of T. Furthermore, the vertices in $V_u = V(T_u) \setminus \{u\}$ (all the children of u) have exactly one neighbor in W and $1 \leq |V_u| \leq 2$. Also, either $V(T) \setminus V(T_u) = \emptyset$ or the parent x of u is in $\mathcal{R}$. To prove the lemma, we will show that if $(G, W, \mathcal{R}, k)$ is a Yes instance of

DIS-IFVS then there is an ifvs, $S \subseteq V(H) \setminus (W \cup \mathcal{R})$, of size at most k in G such that $S \cap V_u = \emptyset$. Observe that x (if it exists) cannot belong to S.

Let $S \subseteq V(H) \setminus (W \cup \mathcal{R})$ be an ifvs in G of size at most k. If $S \cap V_u = \emptyset$ then S is the desired solution. Otherwise, let $S' = (S \setminus V_u) \cup \{u\}$. Since $S \cap V_u \neq \emptyset$, we have that u does not belong to S and thus the size of S' is also at most k. We claim that S' is an ifvs of the desired form. Notice that $S' \subseteq V(H) \setminus (W \cup \mathcal{R})$ holds. Also, S' is an independent set since neighbors of u do not belong to S' and $S \setminus V_u$ is an independent set. Therefore, we only need to prove that S' is a feedback vertex set in G. Suppose not, then there is a cycle C in $G - S'$. If C does not contain any vertex from $V_u \cup \{x\}$, then C is also a cycle in $G - S$, contradicting that S in an ifvs in G. If C contains x, but does not contain any other vertex from V_u, then we can conclude that C is a cycle in $G - S$, since $x \notin S$. Otherwise, C contain a vertex say, $v \in V_u$. Note that v is a degree 2 vertex in G. This implies that any cycle containing v must contain both the neighbors of v. But then u belongs to C contradicting that C is a cycle in $G - S'$. This proves the safeness of the reduction rule.

When we add $V_u \cup \{x\}$ to W the number of components can increase at most by 1. Note that none of the vertices in $V_u \cup \{x\}$ is a tent. Therefore, the number of nice vertices does not decrease and u becomes a tent. This implies that the measure μ does not increase. This concludes the proof. □

Algorithm Description. We give an algorithm only for the decision variant of the problem. It is straightforward to modify the algorithm so that it actually finds a solution, provided there exists one.

We will follow a branching strategy with a non-trivial measure function. Let $(G, W, \mathcal{R}, k)$ be the input instance. The algorithm first applies Reduction Rules 14.2.1-14.2.8, in this order, exhaustively. That is, at any point of time we apply the lowest numbered applicable Reduction Rule. For clarity we denote the reduced instance (the one on which Reduction Rules 14.2.1-14.2.8 do not apply) by $(G, W, \mathcal{R}, k)$.

We now check whether every vertex in $V(G) \setminus (W \cup \mathcal{R})$ is either nice or a tent. If this is the case, then in polynomial time we check whether there is an ifvs contained in $V(G) \setminus (W \cup \mathcal{R})$ of size at most k and return accordingly by Lemma 14.2.6.

Lemma 14.2.6. *Let (G, X) be an instance of* IFVS *where every vertex in $V(G) \setminus X$ is either nice or is a tent. Then in polynomial time we can find a minimum sized ifvs $S \subseteq V(G) \setminus X$ in G.*

The proof of Lemma 14.2.6 follows from Lemma 4.10 in [65], which is based on a polynomial time algorithm for FVS in subcubic graphs by Ueno et at. [202] and the fact that the algorithm described in [65] for finding feedback vertex set on the instances of described type always returns an independent feedback vertex set (if it exists).

Finally, we move to the branching step of the algorithm. We never *branch on a nice vertex or a tent*. We will branch on the vertices in $V(H) \setminus \mathcal{R}$ based on certain criteria. We consider the following three scenarios.

- **Scenario A** – There is a vertex which in not a *tent* and has at least 3 neighbors in W.
- **Scenario B** – There is a leaf which is not a *nice vertex* and has exactly 2 neighbors in W, but no leaf has more than 2 neighbors in W.
- **Scenario C** – All the leaves have exactly one neighbor in W.

Scenario A. If there is a vertex $v \in V(H)$ which is not a *tent* and has at least 3 neighbors in W. Note that $v \notin \mathcal{R}$ as the Reduction Rule 14.2.5 is not applicable. In this case we branch on v as follows.

- When v belongs to the solution, then all the vertices in $N_H(v)$ cannot belong to the solution. Therefore, we add all the vertices in $N_H(v)$ to the set $\mathcal{R}$. The resulting instance is $(G - \{v\}, W, \mathcal{R} \cup N(v), k-1)$. In this case k decreases by 1 and $\rho(W), \eta, \tau$ remains the same. Therefore, μ decreases by 2.
- When v does not belong to the solution, then we add v to W. The resulting instance is $(G, W \cup \{v\}, \mathcal{R}, k)$. Note that v cannot have two neighbors in the same component of W, otherwise Reduction Rule 14.2.6 would be applicable. Therefore, $G[W \cup \{v\}]$ has at most $\rho(W) - 2$ components. Also, k does not change and η, τ does not decrease. Therefore, μ decreases at least by 2.

The resulting branching vector for this case is $(2, 2)$. When none of the Reduction rules are applicable and we cannot branch according to Scenario A, then we can assume that there is no vertex $v \in V(H)$, such that v has more than 2 neighbors in W. Of course a tent could have three neighbors in W but as stated before we *never* branch on a nice vertex or a tent. For each tree T (a connected component) in H, for a vertex $v \in V(T)$ we define the level $\ell(v)$ of v to be the distance of v from the

root of T. The root r in a tree has $\ell(r) = 0$. We call a leaf vertex $v \in V(T)$ as a *deep leaf* if $\ell(v) \neq 0$ and for all leaves $v' \in V(T)$, $\ell(v') \leq \ell(v)$.

Scenario B. Let v be a leaf in some tree T in H with the unique neighbor $u \in V(H)$ such that v has exactly two neighbors in W. Observe that $u \notin \mathcal{R}$ since Reduction Rule 14.2.7 is not applicable. We branch on u as follows.

- When u belongs to the solution, then all the vertices in $N_H(u)$ cannot belong to the solution. We add all the vertices in $N_H(u) \setminus \{v\}$ to the set $\mathcal{R}$. We add the vertex v to W. The resulting instance is $(G - \{u\}, W \cup \{v\}, \mathcal{R} \cup (N_H(u) \setminus \{v\}), k - 1)$. In this case k decreases by 1, and η, τ do not decrease. The number of components in $G[W \cup \{v\}]$ is $\rho(W) - 1$, since v has 2 neighbors in different components of W. Therefore, μ decreases by 3.
- When u does not belong to the solution, then we add u to W. The resulting instance is $(G, W \cup \{u\}, \mathcal{R}, k)$. Note that when we add u to W then v becomes a *tent*. The number of components in $G[W \cup \{u\}]$ is at most $\rho(W) + 1$. Note that k does not increase, η does not decrease and τ increases at least by 1. Therefore, μ decreases by at least 1.

The resulting branching vector for this case is $(3, 1)$.

We now assume that all the leaves in H have exactly one neighbor in W.

Scenario C. Let v be a *deep leaf* in some tree T in H. Let the unique neighbor of v in H be u. We note that the sub-tree T_u rooted at u is a star i.e., u is the only vertex in T_u which can possibly have degree more than one. This follows from the fact that v is a *deep leaf*. Also, $u \notin \mathcal{R}$ since Reduction Rule 14.2.7 is not applicable. We consider the following cases depending on the number of leaves in the sub-tree T_u rooted at u.

Case 1. If T_u has at least two more leaves, say x, y other than v. We branch on the vertex u as follows.

- When u belongs to the solution, then the vertices in $N_H(u)$ does not belong to the solution. We add all the vertices in $N_H(u)$ to the set $\mathcal{R}$. The resulting instance is $(G - \{u\}, W, \mathcal{R} \cup N_H(u), k - 1)$. In this case k decreases by 1 and $\eta, \tau, \rho(W)$ does not change. Therefore, μ decreases at least by 2.
- When u does not belong to the solution, we add u to W. The resulting instance is $(G, W \cup \{u\}, \mathcal{R}, k)$. Observe that when we add u to W then, v, x, y becomes

nice vertices and the number of components in $G[W \cup \{u\}]$ is at most $\rho(W)+1$. Therefore, μ decreases at least by 2.

The resulting branching vector for this case is $(2, 2)$.

Case 2. If T_u has at most one more leaf other than v. We let x to be the parent of u in T. Note that x exists and $x \notin \mathcal{R}$ because each leaf has exactly one neighbor in W and Reduction Rules 14.2.2 and 14.2.8 are not applicable. In this case we branch on x.

- When x belongs to the solution, then the vertices in $N_H(x)$ does not belong to the solution. We add all the vertices in $N_H(x) \setminus \{u\}$ to the set $\mathcal{R}$ and add u to the set W. The resulting instance is $(G-\{x\}, W \cup \{u\}, \mathcal{R} \cup (N_H(x) \setminus \{u\}), k-1)$. Observe that Reduction Rule 14.2.2 is not applicable. Therefore, at least one of the following holds.

  —u has a neighbor in W.

  —u has one more leaf v' other than v.

  In the former case, when we add u to W, the number of components in $G[W \cup \{u\}]$ is at most $\rho(W)$. Also, v becomes a *nice vertex*. Therefore, η increases at least by 1 and τ does not decrease. Therefore, μ decreases at least by 3. In the latter case when we add u to W, v, v' becomes *nice vertices*. In this case k decreases by 1, η increases by 2, τ does not decrease, and $\rho(W)$ can increase at most by 1. Therefore, μ decreases at least by 3.

- When x does not belong to the solution, we add x to W. But then T_u is a star and u does not have a parent. Therefore, we can apply the Reduction Rule 14.2.8. That is, we can add v, v' to W. The resulting instance would be $(G, W \cup \{x, v, v'\}, \mathcal{R}, k)$. Observe that u becomes a *tent*. In this case k, ρ remains the same, while τ increases by 1 and $\rho(W)$ can increase at most by 1. Therefore, μ decreases at least by 1.

The resulting branching vector for this case is $(3, 1)$.

This completes the description of the algorithm.

Analysis and Correctness of the Algorithm. The following Lemma which will be used to prove the correctness of the algorithm.

Scenario	Cases	Branch Vector	c^{μ}
Scenario A		$(2,2)$	1.4142^{μ}
Scenario B		$(3,1)$	1.4656^{μ}
Scenario C	Case 1	$(2,2)$	1.4142^{μ}
	Case 2	$(3,1)$	1.4656^{μ}

Table 14.1: The branch vectors and the corresponding running times.

Lemma 14.2.7. *For an instance $I = (G, W, \mathcal{R}, k)$ of* Dis-IFVS*, if $\mu < 0$, then I is a* No*instance.*

Proof. Let us assume for contradiction that I is a Yes instance and $\mu < 0$. Let $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ be an ifvs in G of size at most k. Therefore, $F = G - S$ is a forest. Let $N \subseteq V(G) \setminus (W \cup \mathcal{R})$, $T \subseteq V(G) \setminus (W \cup \mathcal{R})$ be the set of nice vertices and tents in $V(G) \setminus (W \cup \mathcal{R})$, respectively. Since F is a forest we have that $G' = G[(W \cup N \cup T) - S]$ is a forest. In G', we contract each of the connected components in W to a single vertex to obtain a forest $\tilde{F}$. Observe that $\tilde{F}$ has at most $|V(\tilde{F})| \leq \rho(W) + |N \setminus S| + |T \setminus S|$ vertices and thus can have at most $\rho(W)+|N\setminus S|+|T\setminus S|-1$ many edges. The vertices in $(N\cup T)\setminus S \subseteq V(G)\setminus(W\cup\mathcal{R})$ forms an independent set in $\tilde{F}$, since they are nice vertices or tents. The vertices in $N \setminus S$ and $T \setminus S$ have degree 2 and degree 3 in $\tilde{F}$, respectively, since their degree cannot drop while contracting the components of $G[W]$. This implies that,

$$2|N \setminus S| + 3|T \setminus S| \leq |E(\tilde{F})| \leq \rho(W) + |N \setminus S| + |T \setminus S| - 1.$$

Therefore, $|N \setminus S| + 2|T \setminus S| < \rho(W)$. But $N \cap T = \emptyset$ and thus

$$|N| + 2|T| < \rho(W) + 2|S| \leq \rho(W) + 2k. \tag{14.1}$$

However, by our assumption, $\mu(I) = \rho(W) + 2k - (|N| + 2|T|) < 0$ and thus $|N| + 2|T| > \rho(W) + k$. This, contradicts the inequality given in Equation 14.1 contradicting our assumption that I is a Yes instance.

□

Lemma 14.2.8. *The algorithm presented for* Dis-IFVS *is correct.*

Proof. Let $I = (G, W, \mathcal{R}, k)$ be an instance of Dis-IFVS. We prove the correctness of the algorithm by induction on $\mu = \mu(I) = 2k + \rho(W) - (\eta + 2\tau)$. The base case occurs in one of the following cases.

- $\mu < 0$. By Lemma 14.2.7, when $\mu < 0$, we can correctly conclude that I is a No instance.

- $k < 0$. By Reduction Rule 14.2.3 it follows that when $k < 0$, we can correctly conclude that I is a No instance.

- When none of the Reduction Rules and Branching Rules are applicable. In this case we are able to solve the instance in polynomial time.

By induction hypothesis we assume that for all $\mu \leq l$, the algorithm is correct. We will now prove that the algorithm is correct when $\mu = l + 1$. The algorithm does one of the following. Either applies one of the Reduction Rules if applicable. By Lemma 14.2.1 to Lemma 14.2.5 we know that the Reduction Rules correctly concludes that I is a No instance or produces an equivalent instance I' with $\mu(I') \leq \mu(I)$. If $\mu(I') < \mu(I)$, then by induction hypothesis and safeness of the Reduction Rules the algorithm correctly decides if I is a yes instance or not. Otherwise, $\mu(I') = \mu(I)$. If none of the Reduction Rules are applicable then the algorithm applies one of the Branching Rules. Branching Rules are exhaustive and covers all possible cases. Furthermore, μ decreases in each of the branch by at least one. Therefore, by the induction hypothesis, the algorithm correctly decides whether I is a Yes instance or not. □

Theorem 14.2.1. *The algorithm presented solves* Disjoint Independent Feedback Vertex Set *in time* $\mathcal{O}^\star(3.1481^k)$.

Proof. The Reduction Rules 14.2.1 to 14.2.8 can be applied in time polynomial in the input size. Also, at each of the branch we spend a polynomial amount of time. At each of the recursive calls in a branch, the measure μ decreases at least by 1. When $\mu \leq 0$, then we are able to solve the remaining instance in polynomial time or correctly conclude that the corresponding branch cannot lead to a solution. At the start of the algorithm $\mu \leq 3k$. The worst-case branching vector for the algorithm is $(3, 1)$ (see Table 14.1). The recurrence for the worst case branching vector is:

$$T(\mu) \leq T(\mu - 3) + T(\mu - 1)$$

The running time corresponding to the above recurrence relation is 3.1481^k. □

14.2.2 A Family of Counter Examples to Song's Algorithm for IFVS

A fundamental problem in Song's algorithm [196] is the following. As a consequence of the technique of iterative compression, in a typical case (ignoring some details that are irrelevant here) the problem reduces to its disjoint version where additionally given a feedback vertex set F of the input graph G, the goal is to find a solution that is disjoint from F. In this situation, say $T_1, \ldots, T_p$ are the trees of the forest of $G - F$, the author assumes that it is enough to look for a disjoint (from F) solution in $G[F \cup T_i]$, for each $i \in [p]$ and return the union of these solutions as the final solution. This assumption is blatantly wrong as solutions obtained this way do not kill cycles that pass through more than one tree T_i. This is formalized below where we give a family of counter examples to Song's algorithm.

Let $\mathcal{F}$ be the family of even cycles. For any $C \in \mathcal{F}$, let (C_W, C_H) be a bipartition of C. Given a graph G and a feedback vertex set F in G, Lemma 3.1 of [196] claims to output a minimum IFVS in G. But for $G = C$ and $F = C_W$, where $C \in \mathcal{F}$, the algorithm of Lemma 3.1 always returns ϕ.

14.3 Exact Algorithm for Independent Feedback Vertex Set

In this section we give an exact algorithm for IFVS running in time $\mathcal{O}^\star(1.5981^n)$ using the technique of measure and conquer. To solve IFVS, in fact we will solve the following generalization of IFVS.

Excluded-Independent Feedback Vertex Set (Exc-IFVS)
Input: An undirected (multi) graph G, a set $M \subseteq V(G)$
Output: A minimum sized set $S \subseteq V(G) \setminus M$ such that S is an independent set and $G - S$ is a forest, if it exists. Otherwise, return ∞.

Observe that IFVS is a special case of Exc-IFVS when $M = \emptyset$. Given a graph G and a set M, a subset $S \subseteq V(G) \setminus M$ is called an *M-ifvs*, if S is an independent set and $G - S$ is a forest.

The algorithm for Exc-IFVS is recursive and apart from branching on a care-

fully selected vertex it applies Reduction Rules 14.2.1 and 14.2.2, described in Section 14.2. We note that when applying Reduction Rule 14.2.2 we prefer deleting a vertex in M and adding an edge between its neighbours. Along with Reduction Rules 14.2.1 and 14.2.2, we will also need the following two new reduction rules for our exact algorithm.

Reduction Rule 14.3.1. *If $G[M]$ contains a cycle then return ∞.*

The correctness of the Reduction Rule 14.3.1 follows from the fact that if $G[M]$ contains a cycle then there is no M-ifvs and thus we return ∞.

Reduction Rule 14.3.2. *If there is $(u, v) \in E(G)$ such that $u, v \in M$, then contract the edge (u, v). Let $v^\star$ be the vertex obtained after contracting (u, v) in G. The resulting instance is $(G/(u, v), (M \setminus \{u, v\}) \cup \{v^\star\})$.*

Observe that there is an one-to-one correspondence between cycles in G and G/e; which implies the correctness of the Reduction Rule 14.3.2. A more formal argument is presented in the next lemma.

Lemma 14.3.1. *Reduction Rule 14.3.2 is safe.*

Proof. Let $(u, v) \in E(G)$ such that $u, v \in M$. Let $G^\star = G/e$ and $v^\star \in V(G^\star)$ be the vertex obtained after contracting (u, v) in G. We will show that (G, M) has an ifvs $S \subseteq V(G) \setminus M$ of size k if and only if $(G^\star, (M \setminus \{u, v\}) \cup \{v^\star\})$ has an ifvs $S' \subseteq V(G/e) \setminus M$ of size k.

In the forward direction let $S \subseteq V(G) \setminus M$ be an ifvs in G such that $|S| \leq k$. Observe that $S \subseteq V(G^\star) \setminus M$ and S is an independent set in $G^\star$. We will show that S is an fvs in $G^\star$. Suppose not, then there is a cycle C in $G^\star$. If C does not contain $v^\star$ then C is a cycle in G, contradicting that S is an ifvs in G. Therefore, we assume that C contains $v^\star$. Let x, y be neighbours of $v^\star$ in C. If $x, y \in N_G(u)$ then, we get a cycle C' in G by replacing $v^\star$ by u. A similar argument can be given when $x, y \in N_G(v)$. Otherwise, exactly one of x, y is a neighbour of u in G and the other is a neighbour of v. Therefore, we can replace $v^\star$ in C by the edge (u, v) and obtain a cycle in G, contradicting that S was an ifvs in G.

In the reverse direction consider an ifvs $S^\star$ in $G^\star$. We claim that $S^\star$ is an ifvs in G. Observe that $S^\star \subseteq V(G) \setminus M$ and $S^\star$ is an independent set in G. We only need to prove that $S^\star$ is an fvs in G. Suppose not, then there is a cycle C in $G - S^\star$. If none of $\{u, v\}$ belongs to the cycle C then, C is also a cycle in $G^\star$, contradicting

the assumption that $S^\star$ is an ifvs in $G^\star$. Hence, C contains at least one of $\{u, v\}$. Suppose that C either contains exactly one of $\{u, v\}$ or it contains the edge (u, v). Then we get a cycle in $G^\star$ by replacing u (or v) or the edge (u, v) by $v^\star$. Suppose that C contains both $\{u, v\}$ but not the edge (u, v). In this case, C has a path P from u to v (which is not an edge). But then, $V(P) \setminus \{u, v\}$ along with $v^\star$ forms a cycle in $G^\star$, contradicting the assumption that $S^\star$ is an ifvs in $G^\star$. Hence the result is proved. □

In fact, the proof of Lemma 14.3.1 implies the following result which will be used crucially in the next section when we upper bound the number of minimal ifvses in a graph on n vertices.

Lemma 14.3.2. *Let G be a graph, $M \subseteq V(G)$ and $e = (u, v) \in E(G)$ with $u, v \in M$. Furthermore, let $G^\star = G/e$, $v^\star \in V(G^\star)$ be the vertex obtained after contracting (u, v) and $M^\star = M \setminus \{u, v\} \cup \{v^\star\}$. Then a set $S \subseteq V(G) \setminus M$ is a minimal M-ifvs in G if and only if S is a minimal $M^\star$-ifvs in $G^\star$.*

We start with a few definitions. Let (G, M) be an instance of Exc-IFVS and $U = V(G) \setminus M$. Recall that, a vertex $v \in U$ is called a *tent* if its degree in G is 3 and all its neighbours are in M. In other words, v has degree 3 in G and is an isolated vertex in $G[U]$. We call a vertex $v \in U$ *nice* if its degree in G is 2 and both of its neighbours are in M. Lemma 14.2.6 explains our interest in nice vertices and tents.

Lemma 14.3.3. Exc-IFVS *can be solved in time $\mathcal{O}^\star(1.5981^n)$.*

Proof. We give an algorithm for the variant of the problem, where we return *the size* of a minimum sized ifvs, if it exists. It is straightforward to modify the algorithm so that it actually finds a solution, provided that one exists.

We will follow a branching strategy with a nontrivial measure function. Let (G, M) be the input instance. The algorithm first applies Reduction Rules 14.2.1, 14.2.2, 14.3.1 and 14.3.2, in that order, exhaustively. That is, at any point of time we apply the lowest numbered applicable Reduction Rule. For simplicity of notation, we denote the reduced instance (the one on which Reduction Rules 14.2.1, 14.2.2, 14.3.1 and 14.3.2 do not apply) by (G, M).

We then check whether every vertex in $V(G) \setminus M$ is either nice or a tent. If this is the case, we apply Lemma 14.2.6 and solve the problem in polynomial time.

Otherwise, we move to the branching step of the algorithm. We will call a vertex $v \in V(G) \setminus M$ *branchable* if either $d_G(v) \geq 4$ or, $d_G(v) = 3$ and there exist $u \in N(v)$

degree of v	u_v	m_v	Worst case branching vector	Running time (c^μ)
degree of $v = 3$	1	2	$(67, 56)$	1.01137^μ
	2	1	$(93, 41)$	1.011^μ
	3	0	$(119, 26)$	1.01149^μ
degree of $v \geq 4$	0	≥ 4	$(41, 86)$	1.0115^μ
	1	≥ 3	$(67, 71)$	1.0101^μ
	2	≥ 2	$(93, 56)$	1.00956^μ
	3	≥ 1	$(119, 41)$	1.00955^μ
	≥ 4	≥ 0	$(145, 26)$	1.01016^μ

Table 14.2: The branch vectors and the corresponding running times.

such that $u \notin M$. The algorithm has only one branching rule which is as follows: *Pick a branchable vertex v of largest degree and branch.*

We branch by including v in the solution in one branch, and excluding it in the other branch. That is, we call the algorithm on instances $(G - \{v\}, M \cup N(v))$ and $(G, M \cup \{v\})$. Let α and β be the values returned by the algorithm when run on the instances $(G - \{v\}, M \cup N(v))$ and $(G, M \cup \{v\})$, respectively. Finally, we return

$$\min\{\alpha + 1, \beta\}.$$

The correctness of this algorithm follows from the safeness of reductions and the fact that the branching is exhaustive.

To estimate the running time of the algorithm, for instance $I = (G, M)$, we define its measure $\mu = \mu(G, M) = 15|M| + 41|U|$, where $U = V(G) \setminus M$. Let $m_v = |N(v) \cap M|$ and $u_v = |N(v)| - m_v$. Branching on v leads to following cases.

- When v belongs to the solution then none of the vertices in $N(v)$ belong to the solution. Therefore, we add all the vertices in $N(v)$ to the set M. The resulting instance is $(G-\{v\}, M\cup N(v))$. In this case $|U|$ decreases by at least u_v+1 and $|M|$ increases by u_v. Consequently, μ decreases by $41(u_v+1)-15u_v = 26u_v+41$.
- When v does not belong to the solution, we add v to M. Observe that since Reduction Rule 14.3.2 is not applicable, $N(v) \cap M$ is an independent set in G.

  Thus, once we add v to M Reduction Rule 14.3.2 applies $|N(v) \cap M|$ times. Each application of Reduction Rule 14.3.2 decreases $|M|$ by 1. We note that when we add v to M, $|M|$ increased by 1. Therefore, at this branch after exhaustive application of Reduction Rule 14.3.2, $|M\cup\{v\}|$ decreases by m_v-1. Also, $|U|$ decreases by 1. Therefore, the measure μ decreases by $26 + 15m_v$

The resulting branching vector for this case is $(26u_v + 41, 26 + 15m_v)$.

Corresponding to the different values that u_v and m_v can take, the exact branching vector and the corresponding running time is given in Table 14.2. The worst case branching vector is given by $d_G(v) = 4$ and $m_v = 4$. The recurrence relation corresponding to the worst case branching vector is the following :

$$T(\mu) \leq T(\mu - 41) + T(\mu - 86).$$

The running time obtained from the above recurrence relation is 1.5981^n. To upper bound the running time of the algorithm, we now bound the number of times any reduction rule is applied and the amount of time it takes to apply any reduction rule. It is evident that each of the reduction rule can be applied in polynomial time. Furthermore, each time a reduction rule is applied either the number of vertices or the number of edges in the newly obtained instance drops by 1 or we conclude that it is a no instance. Hence, the reduction rules can be applied at most $\mathcal{O}(n^2)$ times. Thus the total number of nodes in the search tree is upper bounded by $\mathcal{O}^\star(1.5981^n)$ and on each node the algorithm takes $n^{\mathcal{O}(1)}$ time, resulting in the desired running time for the algorithm. □

As an immediate corollary of Lemma 14.3.3 we obtain the following theorem.

Theorem 2. IFVS *can be solved in time* $\mathcal{O}^\star(1.5981^n)$.

14.4 On the Number of Minimal Independent Feedback Vertex Sets

In this section we use the method of measure and conquer (as in the previous section) to obtain an upper bound of 1.7480^n for the number of *minimal independent feedback vertex sets (minimal-ifvs)* in a graph G on n vertices. We start by defining some auxiliary notions that will be useful for our purpose.

Definition 14.4.1. *For a graph G, and a subset $M \subseteq V(G)$, we define $\mathcal{I}_G(M)$ to be the set of all minimal-ifvs S in the graph G such that $S \subseteq V(G) \setminus M$. In other words, S is a minimal M-ifvs. We will refer to M as the marked set.*

Thus, the problem of upper bounding the number of minimal-ifvses in G can be stated as finding an upper bound on $|\mathcal{I}_G(\emptyset)|$.

For a graph G, and a subset $M \subseteq V(G)$, to upper bound the size of $\mathcal{I}_G(M)$, we will give an inductive proof which mimics the recursive algorithm described in Lemma 14.3.3. The proof will also use Reduction Rules 14.2.1 and 14.3.2. For our proof we also need an upper bound on the number of minimal-ifvses of a special instance.

Definition 14.4.2. *An instance $I = (G, M)$ is called* special *if (a) G is connected; (b) every vertex $v \in V(G)$ has degree at least* 2*; (c) every vertex $v \in (U = V(G) \setminus M)$ has degree* exactly 2*; and (d) the sets M and U are independent. Let G^{sp} denote the graph with vertex set M and edge set $E(G^{\mathsf{sp}})$, where for every vertex $v \in U$, there is an edge e_v between the neighbors of v. Observe that there is an one-to-one correspondence between edges in $E(G^{\mathsf{sp}})$ and vertices in U (we keep multi-edges).*

In the following lemmas we characterize minimal-ifvses of a special instance and we upper bound the size of $\mathcal{I}_G(M)$. For a special instance (G, M) and $S \subseteq V(G)$ by E_S we will denote the set of edges in G^{sp} corresponding to the vertices in S and $U = V(G) \setminus M$.

Lemma 14.4.1. *Let $I = (G, M)$ be a special instance. Then G is a bipartite graph, such that $|U| \geq |M|$.*

Proof. The assertion that G is a is a bipartite graph follows from the definition of special instance. Next we show that $|U| \geq |M|$. Note that every vertex in U must have degree exactly 2. But a vertex in M has degree at least 2. Since $\sum_{u \in U} d_G(u) = \sum_{v \in M} d_G(v)$, we conclude that $2|U| = \sum_{v \in M} d_G(v) \geq 2|M|$. Thus, $|U| \geq |M|$, is proved. □

Lemma 14.4.2. *Let $I = (G, M)$ be a special instance and $S \subseteq U$. Then S is a M-ifvs in G if and only if $G^{\mathsf{sp}} - E_S$ is a forest.*

Proof. In the forward direction let S be a M-ifvs in G. We show that $G^{\mathsf{sp}} - E_S$ is acyclic. Suppose not, then there is a cycle $C = (m_0, m_1, \ldots, m_\ell)$ in $G^{\mathsf{sp}} - E_S$. An edge $(m_i, m_{(i+1) \mod (\ell+1)})$, $0 \leq i \leq \ell$, corresponds to a degree 2 vertex u_i in U with neighbours $m_i, m_{(i+1) \mod (\ell+1)}$ in G. Therefore, $(m_0, u_0, m_1, u_2, \ldots, m_\ell, u_\ell)$ forms a cycle in G, contradicting that S is a M-ifvs in G. In the reverse direction suppose $G^{\mathsf{sp}} - E_S$ is a forest and $G - S$ contains a cycle say C. But then if we delete the vertices of U from C, and add edges between its neighbours, we obtain a cycle in $G^{\mathsf{sp}} - E_S$, contradicting that it was a forest. This concludes the proof. □

Lemma 14.4.3. *Let $I = (G, M)$ be a special instance and S be a minimal M-ifvs in G. Then, the edges corresponding to $U \setminus S$ forms a spanning tree in G^{sp}.*

Proof. We first note that since G is connected, G^{sp} is also connected. From Lemma 14.4.2, it follows that $G^{\mathsf{sp}} - E_S$ is a forest. We show that $G^{\mathsf{sp}} - E_S$ is connected. Assume otherwise, then $G^{\mathsf{sp}} - E_S$ has at least two connected components. Let $\mathcal{C}$ be one of the connected components in $G^{\mathsf{sp}} - E_S$. Observe that $V(\mathcal{C})$ and $M \setminus V(\mathcal{C}) \neq \emptyset$ is a partition of vertices in M and there is at least one vertex in U which has a neighbour in $V(\mathcal{C})$ as well as $M \setminus V(\mathcal{C})$. Let X be the set of vertices in U which have a neighbour in $V(\mathcal{C})$ as well as $M \setminus V(\mathcal{C})$. Since, $G^{\mathsf{sp}} - E_S$ is not connected, therefore S must contain a vertex $x \in X$. But then, $G^{\mathsf{sp}} - (E_S \setminus \{x\})$ is a forest. From Lemma 14.4.2, it implies $S \setminus \{x\}$ is also a M-ifvs in G, contradicting the minimality of S. □

Lemma 14.4.4. *Let $I = (G, M)$ be a special instance. Then every minimal M-ifvs in G is also a minimum M-ifvs with size $|U| - (|M| - 1)$.*

Proof. Observe that the number of edges in G^{sp} is $|U|$. From Lemmata 14.4.1 and 14.4.3 it follows that $|S| = |U| - (|M| - 1)$. Therefore, every minimal M-ifvs is a minimum M-ifvs in G with size $|U| - (|M| - 1)$. □

Lemma 14.4.5. *Let $I = (G, M)$ be a special instance on an n-vertex graph. Then, $|\mathcal{I}_G(M)| \leq \left(\frac{2\alpha}{1-\alpha}\right)^{(1-\alpha)n}$, where $|M| = (1-\alpha)n$ and $\frac{1}{2} \leq \alpha \leq 1$.*

Proof. Our assumption implies that $|U| = \alpha n$, where $\frac{1}{2} \leq \alpha \leq 1$. Thus, we have that $|M| = (1-\alpha)n$. By Lemma 14.4.4, the number of minimal (which is same as minimum) M-ifvs is same as the number of spanning trees of G^{sp}. Let $\zeta(H)$ denote the number of spanning trees of a graph H. Grimmett [111] showed that for a connected graph H on n^* vertices and m^* edges

$$\zeta(H) \leq \frac{1}{n}\left(\frac{2m^*}{n^*-1}\right)^{n^*-1}.$$

For $n^* \geq 2$ one can easily show that

$$\zeta(H) \leq \frac{1}{n}\left(\frac{2m^*}{n^*-1}\right)^{n^*-1} \leq \left(\frac{2m^*}{n^*}\right)^{n^*}. \tag{14.2}$$

If $n = 1$ then $|\mathcal{I}_G(M)| \leq 1$, so from now onwards we assume that $n \geq 2$. By

Equation 14.2 we know that the

$$|\mathcal{I}_G(M)| \leq \left(\frac{2\alpha n}{(1-\alpha)n}\right)^{(1-\alpha)n} \leq \left(\frac{2\alpha}{1-\alpha}\right)^{(1-\alpha)n}.$$

This completes the proof. □

Lemma 14.4.6. *An instance $I = (G, M)$ contains at most 1.7480^n minimal M-ifvs, where $n = |V(G)|$. That is, $|\mathcal{I}_G(M)| \leq 1.7480^n$.*

Proof. To estimate the upper bound on $|\mathcal{I}_G(M)|$, for instance $I = (G, M)$, we define its measure $\mu = \mu(G, M) = 33|M| + 67|U|$, where $U = V(G) \setminus M$. Let $f(G, M) = |\mathcal{I}_G(M)|$ and let $f(\mu)$ be a maximum $f(G, M)$ among all pairs (G, M) of measure at most μ. We claim that for $x = 1.00837$, $f(\mu) \leq x^\mu$. We will prove the claim using induction on μ. Observe that for $\mu \leq 67$, either the graph has exactly one vertex in U or has at most 2 vertices in M. Thus, $f(\mu) \leq 1$ and the claim holds. Thus, by induction hypothesis assume that $f(\ell) \leq x^\ell$ for every $\ell < \mu$. Let $I = (G, M)$ be an instance of measure μ.

Our proof is based on several cases and we apply the first applicable case. That is, when we are in Case i, we assume that all the cases with index strictly less than i are not applicable.

Case 1: G is not connected. Let $G_1, \ldots, G_\ell$ denote the connected components of G. Furthermore, let M_i denotes $V(G_i) \cap M$, $i \in \{1, \ldots, \ell\}$. Observe that $\mu(G_i, M_i) < \mu$ and any minimal M-ifvs is formed by the union of minimal M_i-ifvs of G_i, respectively. Thus, by the induction hypothesis we have the following:

$$f(\mu) = \prod_{i=1}^{\ell} f(G_i, M_i) \leq \prod_{i=1}^{\ell} x^{\mu(G_i, M_i)} \leq x^{\sum_{i=1}^{\ell} \mu(G_i, M_i)} = x^\mu.$$

Case 2: Reduction Rules 14.2.1 and 14.3.2 are applicable. Suppose Reduction Rule 14.2.1 applies on a vertex v. Then, since v has degree one, it does not participate in any cycle in G and thus $\mathcal{I}_G(M) = \mathcal{I}_{G\setminus\{v\}}(M \setminus \{v\})$. Furthermore, $\mu(G \setminus \{v\}, M \setminus \{v\}) < \mu$ and thus by induction hypothesis we have that

$$f(\mu) = f(G \setminus \{v\}, M \setminus \{v\}) \leq x^{\mu(G\setminus\{v\}, M\setminus\{v\})} \leq x^\mu.$$

Now let us assume that Reduction Rule 14.3.2 applies on $e = (u, v) \in E(G)$ such that $u, v \in M$. In this case we contract the edge (u, v) to a vertex $v^\star$.

degree of v	u_v	m_v	Worst case branch vector	Running time (c^μ)
degree of $v = 2$	1	1	$(134, 67)$	1.00721^μ
	2	0	$(201, 34)$	1.00748^μ
degree of $v \geq 3$	0	≥ 3	$(67, 133)$	1.00724^μ
	1	≥ 2	$(101, 100)$	1.00692^μ
	2	≥ 1	$(135, 67)$	1.00718^μ
	≥ 3	≥ 0	$(169, 34)$	1.00834^μ

Table 14.3: The branch vectors and the corresponding upper bound on the number of minimal minimal M-ifvs.

Let $G^\star = G/e$ and $M^\star = M \setminus \{u, v\} \cup \{v^\star\}$. Then, by Lemma 14.3.2, we have that $\mathcal{I}_G(M) = \mathcal{I}_{G^\star}(M^\star)$. Observe that $\mathcal{I}_G(M) = \mathcal{I}_{G^\star}(M^\star)$. Furthermore, $\mu(G^\star, M^\star) < \mu$. Now similar to the case of Reduction Rule 14.2.1, one can show that $f(\mu) \leq x^\mu$.

Case 3: Branchable Vertex. We will call a vertex $v \in U$ *branchable* if either $d_G(v) \geq 3$ or, or $d_G(v) = 2$, and there exists $u \in N(v)$ such that $u \notin M$. For an upper bound we proceed as follows: *Select a branchable vertex v of largest degree.* Every, minimal M-ifvs either contains v, or does not. Thus, the number of minimal M-ifvses of G is equal to the number of minimal M-ifvses of G containing v, that is $f(G \setminus \{v\}, M \cup N(v))$, plus the number of minimal M-ifvses of G not containing v, that is $f(G, M \cup \{v\})$. Let $m_v = |N(v) \cap M|$ and $u_v = |N(v)| - m$. This case analysis on v leads to the following sub-cases. We first deal with the case when $d_G(v) \geq 3$.

- **Case A:** When v belongs to the solution then none of the vertices in $N(v)$ belong to the solution. Therefore, we add all the vertices in $N(v)$ to the set M. The resulting instance is $(G \setminus \{v\}, M \cup N(v))$. In this case $|U|$ decreases by at least $u_v + 1$ and $|M|$ increases by u_v. Therefore, μ decreases by $67(u_v + 1) - 33u_v = 34u_v + 67$.
- **Case B:** When v does not belong to the solution, we add v to M. Observe that since Reduction Rule 14.3.2 is not applicable, we have that $N(v) \cap M$ is an independent set in G. Thus, once we add v to M, Reduction Rule 14.3.2 applies $|N(v) \cap M|$ times. Each application of Reduction Rule 14.3.2 decreases $|M|$ by 1. We note that when we add v to M, $|M|$ increased by 1. Therefore, at this branch after exhaustive application of Reduction Rule 14.3.2, $|M \cup \{v\}|$ decreases by $m_v - 1$. Also, $|U|$ decreases by 1. Therefore, the measure μ decreases by $34 + 33m_v$

The resulting branching vector for this case is $(34u_v + 67, 34 + 33m_v)$.

When $d_G(v) = 2$, then we know that there is no vertex in U, that has degree at least three. Thus, in this case when we include v in the perspective solution, that is when we are in **Case A**, the neighbors of v that are present in U become degree one and hence Reduction Rule 14.2.1 removes them. Thus when $d_G(v) = 2$, the branching vector is $(67(1 + u_v), 34 + 33m_v)$.

Corresponding to the different values u_v and m_v can take, the exact branching vector and the corresponding upper bound is given in Table 14.3.In every case we have $f(\mu) \leq x^{\mu}$.

Case 4: Special Instance. Observe that since Cases $1 - 3$ do not apply, the instance $I = (G, M)$ is special. Let $|U| = \alpha n$, where $\frac{1}{2} \leq \alpha \leq 1$. Then, the measure $\mu(G, M) = 67\alpha n + 33(1 - \alpha)n = n(34\alpha + 33)$. Thus,

$$f(\mu) \leq \max_{\frac{1}{2} \leq \alpha \leq 1} \left\{ \left(\frac{2\alpha}{1-\alpha} \right)^{(1-\alpha)n} \right\} \leq \max_{\frac{1}{2} \leq \alpha \leq 1} \left\{ \left(\frac{2\alpha}{1-\alpha} \right)^{\frac{(1-\alpha)\mu}{34\alpha+33}} \right\}$$

For $0.5 \leq \alpha \leq 1$, the function $\left(\frac{2\alpha}{1-\alpha}\right)^{\frac{(1-\alpha)}{34\alpha+33}}$ achieves its maximum at $\alpha = 0.641541$ and the corresponding value is upper bounded by 1.00837. Thus, in this case also we have that $f(\mu) \leq x^{\mu}$.

By the above case analysis we get that $f(\mu) \leq x^{\mu} \leq 1.00837^{67n} \leq 1.7480^n$. Observe that since spanning trees of a graph can be enumerated in a polynomial delay, we have that our upper bound proof is algorithmic. That is, in time $\mathcal{O}^*(1.7480^n)$ one can enumerate all M-ifvses of G. This concludes the proof. □

As an immediate corollary of Lemma 14.4.6 we get the following theorem.

Theorem 14.4.1. *A graph G on n vertices has at most 1.7480^n minimal ifvses.*

Chapter 15

Independent Set Covering Family-Recycling Algorithms for Independent Vertex Deletion Problems

In this chapter, we present a new combinatorial tool for designing parameterized algorithms. It is a simple linear time randomized algorithm that given as input a d-degenerate graph G and an integer k, outputs an independent set Y, such that for every independent set X in G of size at most k, the probability that X is a subset of Y is at least $\left(\binom{k(d+1)}{k} \cdot k(d+1)\right)^{-1}$. Here, an *independent set* in a graph G is a vertex set X such that no two vertices in X are connected by an edge, and the *degeneracy* of an n-vertex graph G is the minimum integer d such that there exists an ordering $\sigma : V(G) \to \{1, \ldots, n\}$ such that every vertex v has at most d neighbors u with $\sigma(u) > \sigma(v)$. Such an ordering σ is called a *d-degeneracy sequence* of G. We say that a graph is d-degenerate, if G has a d-degeneracy sequence. More concretely, we prove the following result.

Lemma 15.0.1. *There exists a linear*[1] *time randomized algorithm that given as input a d-degenerate graph G and an integer k, outputs an independent set Y, such that for every independent set X in G of size at most k the probability that X is a subset of Y is at least $\left(\binom{k(d+1)}{k} \cdot k(d+1)\right)^{-1}$.*

Proof. Given G, k and a d-degeneracy sequence σ of G the algorithm sets $p =$

[1]The time is purely linear in terms of k and d too.

$\frac{1}{d+1}$ and colors the vertices of G black or white independently with the following probability : a vertex gets color black with probability p and white with probability $1-p$. The algorithm then constructs the set Y which contains every vertex v, such that v is colored black and all the neighbors u of v with $\sigma(u) > \sigma(v)$ are colored white. We first show that Y is an independent set. Suppose not. Let $u, v \in Y$, such that $\sigma(u) < \sigma(v)$ and $uv \in E(G)$. Since $u \in Y$, by the construction of Y, v has to be colored white. This contradicts that $v \in Y$ because every vertex in Y is colored black.

We now give a lower bound on the probability with which a given independent set X of size at most k is contained in Y. Define Z to be the set of vertices u such that u has a neighbor $x \in X$ with $\sigma(x) < \sigma(u)$. Since every $x \in X$ has at most d neighbors u with $\sigma(x) < \sigma(u)$, it follows that $|Z| \leq kd$. Observe that $X \subseteq Y$ precisely when all the vertices in X are colored black and all the vertices in Z are colored white. This happens with probability

$$p^{|X|}(1-p)^{|Z|} \geq \left(\frac{k}{k(d+1)}\right)^k \cdot \left(\frac{kd}{k(d+1)}\right)^{kd} \geq \left[\binom{(d+1)k}{k} \cdot k(d+1)\right]^{-1}.$$

Here, the last inequality follows from the fact that binomial distributions are centered around their expectation. This concludes the proof. □

Lemma 15.0.1 allows us to reduce many problems with an independence constraint to the same problem without the independence requirement. For an example, consider the following four well-studied problems:

- Minimum s-t Separator: Here, the input is a graph G, an integer k and two vertices s and t, and the task is to find a set S of at most k vertices such that s and t are in distinct connected components of $G - S$. This is a classic problem solvable in polynomial time [104, 198].
- Odd Cycle Transversal: Here, the input is a graph G and an integer k, and the task is to find a set S of at most k vertices such that $G-S$ is bipartite. This problem is NP-complete [58] and has numerous fixed-parameter tractable (FPT) algorithms [187, 144]. For all our purposes, the $\mathcal{O}(4^k \cdot k^{\mathcal{O}(1)} \cdot (n+m))$ time algorithms of Iwata et al. [120] and Ramanujan and Saurabh [184] are the most relevant.
- Multicut: Here, the input is a graph G, a set $T = \{\{s_1, t_1\}, \{s_2, t_2\}, \ldots, \{s_\ell, t_\ell\}\}$ of terminal pairs and an integer k, and the task is to find a set S on at most

k vertices such that for every $i \leq \ell$, s_i and t_i are in distinct connected components of $G - S$. Such a set S is called a *multicut* of T in G. This problem is NP-completeeven for 3 terminal pairs, that is, when $l = 3$ [73], but it is FPT [35, 160] parameterized by k, admitting an algorithm [146] with running time $2^{\mathcal{O}(k^3)} \cdot mn \log n$.

- Directed Feedback Vertex Set: Here, the input is a directed graph D and an integer k, and the task is to find a set S on at most k vertices such that $D - S$ is acyclic. This problem is also NP-complete [124] and FPT [51] parameterized by k, admitting an algorithm [146] with running time $\mathcal{O}(k! \cdot 4^k \cdot k^5 \cdot (n + m))$.

In the "stable" versions of all of the above-mentioned problems, the solution set S is required to be an independent set. Independent sets are sometimes called stable sets in the literature. In this thesis we stick to independent sets, except for problem names, which are inherited from Marx et al. [159].

Lemma 15.0.1 only applies to graphs of bounded degeneracy. Even though the class of graphs of bounded degeneracy is quite rich (it includes planar graphs, and more generally all graphs excluding a topological minor) it is natural to ask whether Lemma 15.0.1 could be generalized to work for all graphs. However, if G consists of k disjoint cliques of size n/k each, the best success probability one can hope for is $(k/n)^k$, which is too low to be useful for FPT algorithms.

The algorithms based on Lemma 15.0.1 are randomized, however they can be derandomized using a new combinatorial object, that we call k-independence covering families, which may be of independent interest. We call an independent set of size at most k a *k-independent set*, and we call a family of independent sets an *independent family*. An independent family $\mathcal{F}$ *covers* all k-independent sets of G, if for every k-independent set X in G there exists an independent set $Y \in \mathcal{F}$ such that $X \subseteq Y$. In this case, we call $\mathcal{F}$ a *k-independence covering family*. An algorithm based on Lemma 15.0.1 can be made deterministic by first constructing a k-independence covering family $\mathcal{F}$, and then looping over all sets $Y \in \mathcal{F}$ instead of repeatedly drawing Y at random using Lemma 15.0.1.

Since a graph G contains at most n^k independent sets of size at most k, drawing $\mathcal{O}(\binom{k(d+1)}{k} \cdot kd \cdot \log n)$ sets using Lemma 15.0.1 and inserting them into $\mathcal{F}$ will result in a k-independence covering family with probability at least $1/2$. Hence, for every d and k, every graph G on n vertices of degeneracy at most d has a k-independence covering family of size at most $\mathcal{O}(\binom{k(d+1)}{k} \cdot kd \cdot \log n)$. By direct applications of existing

pseudo-random constructions (of $(n,(r,s))$-*cover free families*) we show that given a graph G of degeneracy d and integer k one can construct a k-independence covering family of size not larger than $\mathcal{O}(\binom{k(d+1)}{k} \cdot kd \cdot \log n)$ in time roughly proportionate to its size.

Additionally, we also show that for any *nowhere dense* graph class [169, 170], there exists a function f such that given an n-vertex graph from this graph class, any real ϵ and any positive integer k, one can construct a k-independence covering family for this graph of size $f(k,\epsilon) \cdot n^{\epsilon}$. This construction immediately yields FPT algorithms for the considered problems on nowhere dense classes of graphs.

15.1 Independence Covering Family and Lemma

In this section we give constructions of k-independence covering families, which are useful in derandomizing algorithms based on Lemma 15.0.1. Towards this we first formally define the notion of k-independence covering family – a family of independent sets of a graph G which covers all independent sets in G of size at most k.

Definition 15.1.1 (k-Independence Covering Family)**.** *For a graph G and $k \in \mathbb{N}$, a family of independent sets of G is called an independence covering family for (G,k), denoted by $\mathcal{F}(G,k)$, if for any independent set X in G of size at most k, there exists $Y \in \mathcal{F}(G,k)$ such that $X \subseteq Y$.*

Observe that for any pair (G,k), there exists an independence covering family of size at most $\binom{n}{k}$ containing all independent sets of size at most k. We show that, if G has bounded degeneracy, then k-independence covering family of "small" size exists. In fact, we give both randomized and deterministic algorithms to construct such a family of "small" size for graphs of bounded degeneracy. In particular, we prove that if G is d-degenerate, then one can construct an independent set covering family for (G,k) of size $f(k,d) \cdot \log n$, where f is a function depending only on k and d. We first give the randomized algorithm for constructing k-independence covering family. Towards this we use the algorithm described in Lemma 15.0.1. For an ease of reference we present the algorithm given in Lemma 15.0.1 here.

Lemma 15.1.1 (Randomized Independence Covering Lemma)**.** *There is an algorithm that given a d-degenerate graph G and $k \in \mathbb{N}$, outputs a family $\mathcal{F}(G,k)$ such that (a) $\mathcal{F}(G,k)$ is an independence covering family for (G,k) with probability at*

Algorithm 2: Input is (G, k), where G is a d-degenerate graph and $k \in \mathbb{N}$

1 Construct a d-degeneracy sequence σ of G, using Proposition 2.3.1.
2 Set $p = \frac{1}{d+1}$. Independently color each vertex $v \in V(G)$ black with probability p and white with probability $(1-p)$.
3 Let B and W be the set of vertices colored black and white, respectively.
4 $Z := \{v \in B \mid N^f_{G,\sigma}(v) \cap B = \emptyset\}$.
5 **return** Z

least $1 - \frac{1}{n}$, (*b*) $|\mathcal{F}(G,k)| \leq \binom{k(d+1)}{k} \cdot 2k^2(d+1) \cdot \ln n$, *and* (*c*) *the running time of the algorithm is* $\mathcal{O}(|\mathcal{F}(G,k)| \cdot (n+m))$.

Proof. Let $t = \binom{k(d+1)}{k} \cdot k(d+1)$. We now explain the algorithm to construct the family $\mathcal{F}(G,k)$ mentioned in the lemma. We run Algorithm 2 (Lemma 15.0.1) $\gamma = t \cdot 2k \ln n$ times. Let $Z_1, \ldots, Z_\gamma$ be the sets that are output at the end of each iteration of Algorithm 2. Let $\mathcal{F}(G,k)$ be the collection of distinct $Z_i^{'}$s. Clearly, $|\mathcal{F}(G,k)| \leq t \cdot 2k \ln n = \binom{k(d+1)}{k} \cdot 2k^2(d+1) \cdot \ln n$. Thus condition (*b*) is proved. The running time of the algorithm (condition (*c*)) follows from Lemma 15.0.1.

Now we prove condition (*a*) of the lemma. Fix an independent set X in G of cardinality at most k. By Lemma 15.0.1, we know that for any $Z \in \mathcal{F}(G,k)$, $\Pr[X \subseteq Z] \geq \frac{1}{t}$. Thus the probability that there does not exist a set $Z \in \mathcal{F}(G,k)$ such that $X \subseteq Z$ is at most $(1 - \frac{1}{t})^{|\mathcal{F}(G,k)|} \leq e^{-2k \ln n} = n^{-2k}$. The last inequality follows from a well-known fact that $(1-a) \leq e^{-a}$ for any $a \geq 0$. Since the total number of independent sets of size at most k in G is upper bounded by n^k, by the union bound, the probability that there exists an independent set of size at most k which is not a subset of any set in $\mathcal{F}(G,k)$ is upper bounded by $n^{-2k} \cdot n^k = n^{-k} \leq 1/n$. This implies that $\mathcal{F}(G,k)$ is an independence covering family for (G,k) with probability at least $1 - \frac{1}{n}$. □

Remark 15.1.1. *From Fact 2.2.1 and the fact that the number of edges in an n-vertex d-degenerate graph is at most dn, the algorithm of Lemma 15.1.1 runs in time $2^{\mathcal{O}(k \log d)} \cdot n \log n$ and outputs a k-independence covering family of size $2^{\mathcal{O}(k \log d)} \cdot \log n$.*

Deterministic Construction. The deterministic algorithm that we give is obtained from the randomized algorithm presented in Lemma 15.1.1 by using the $(n, (r,s))$-cover free family [37]. The deterministic construction basically replaces the random coloring of the vertices in Line 2 of Algorithm 2 by a coloring defined by a bit string in the $(n, (r,s))$-cover free family. In the following, we first define the $(n, (r,s))$-cover free family, state Proposition 15.1.1 (an algorithm to construct

an $(n,(r,s))$-cover free family of "small" size) which is followed by our deterministic algorithm (Lemma 15.1.2).

Definition 15.1.2 ($(n,(r,s))$-cover free family [37])**.** *Fix positive integers r,s,n with $r,s \leq n$ and let $p := r+s$. An $(n,(r,s))$-cover free family is a set $\mathcal{F} \subseteq \{0,1\}^n$ such that for every $1 \leq i_1 < i_2 < \ldots < i_p \leq n$ and every $J \subset [p]$ of size r, there exists $\mathbf{a} \in \mathcal{F}$ such that $\mathbf{a}_{i_j} = 1$ for all $j \in J$ and $\mathbf{a}_{i_k} = 0$ for all $k \notin J$. Here, $\mathbf{a}_{i_j}$ denotes the i_j-th bit of the bit vector $\mathbf{a}$.*

In the following, for any positive integers r,s and $p = r+s$, the function $N(r,s)$ is defined as $N(r,s) = \frac{p\binom{p}{r}}{\log\binom{p}{r}}$.

Proposition 15.1.1 (Theorem 1, [37])**.** *Fix any integers $r < s < p$ with $p = r+s$. There is an $(n,(r,s))$-cover free family of size $N(r,s)^{1+o(1)} \cdot \log n$ that can be constructed in time $N(r,s)^{1+o(1)} \cdot n \log n$.*

Lemma 15.1.2 (Deterministic Independence Covering Lemma)**.** *There is an algorithm that given a d-degenerate graph G and $k \in \mathbb{N}$, runs in time $N(k,kd)^{1+o(1)}(n+m)\log n$ and outputs a k-independence covering family for (G,k) of size at most $N(k,kd)^{1+o(1)} \cdot \log n$.*

Proof. Let $n = |V(G)|$. Without loss of generality, let $n \geq k(d+1)$, as otherwise the lemma follows trivially. Let us rename the vertex set of the graph to take indices from $[n]$, where $n = |V(G)|$. Let $\mathcal{F}$ be the $(n,(r,s))$-cover free family constructed using Proposition 15.1.1 for $r = k$ and $s = kd$. For each $\mathbf{a} \in \mathcal{F}$, we run Algorithm 2, where Line 2 is replaced as follows: we color the vertex i black if $\mathbf{a}_i = 1$, and white otherwise. More precisely, we run Algorithm 2 for each $\mathbf{a} \in \mathcal{F}$, replacing Line 2 by the procedure just defined, and output the collection $\mathcal{F}(G,k)$ of sets returned at the end of each iteration. The size bound on $|\mathcal{F}(G,k)|$ follows from Proposition 15.1.1 and the running time of the algorithm follows from the fact that each run of Algorithm 2 takes $\mathcal{O}(n+m)$ time.

We now show that $\mathcal{F}(G,k)$ is, indeed, an independent set covering family for (G,k). Let X be an independent set of cardinality at most k in G. Let σ be the d-degenerate sequence constructed in Line 1 of Algorithm 2. Let $Y = \cup_{v \in X} N^f_{G,\sigma}(v)$. Since X is independent, $X \cap Y = \emptyset$. Furthermore, since σ is a d-degeneracy sequence and $|X| \leq k$, we have that $|Y| \leq kd$. If $|X| < k$ (or $|Y| < kd$), then let X' (resp. Y') be a some superset of X (resp. Y) such that $X' \cap Y' = \emptyset$ and, $|X'| = k$, $|Y'| = kd$. Since $n \geq k(d+1)$ such sets X', Y' exist. Let $X' \cup Y' = \{i_1, i_2, \ldots, i_p\}$, where

$p = k(d+1)$ and let $J \subset [p]$ be such that $J = \{j : i_j \in X', j \in [p]\}$. By the definition of $(n,(k,kd))$-cover free family, there is a bit vector $\mathbf{a} \in \mathcal{F}$ such that $\mathbf{a}_{i_j} = 1$ when $i_j \in X'$ and $\mathbf{a}_{i_j} = 0$, when $i_j \in Y'$. Consider the run of Algorithm 2 for the bit vector $\mathbf{a}$. In this run, we have that $X \subseteq B$ and $Y \subseteq W$. From the definition of X, Y and Z (set constructed in Line 4), we have that $X \subseteq Z$. This implies that $\mathcal{F}(G,k)$ is an independence covering family of (G,k). This completes the proof. □

Remark 15.1.2. *From Fact 2.2.1 and the fact that the number of edges in an n-vertex d-degenerate graph is at most dn, the algorithm of Lemma 15.1.2 runs in time $2^{\mathcal{O}(k \log d)} \cdot n \log n$ and outputs a k-independence covering family for (G,k) of size $2^{\mathcal{O}(k \log d)} \cdot \log n$.*

15.1.1 Extensions

For some graphs, whose degeneracy is not bounded, it may still be possible to find a "small" sized independence covering family. This is captured by Corollary 15.1.1.

Corollary 15.1.1. *Let $d, k \in \mathbb{N}$ and G be a graph. Let $S \subseteq V(G)$ be such that $G - S$ is d-degenerate. There is an algorithm which given $d, k \in \mathbb{N}$, G and S, run in time $2^{|S|} \cdot 2^{\mathcal{O}(k \log d)} \cdot (n+m) \log n$ and outputs an independence covering family for (G,k) of size at most $2^{|S|} \cdot 2^{\mathcal{O}(k \log d)} \cdot \log n$*

Proof. Let $G' = G - S$. By the property of S, we know that G' is d-degenerate. We first apply Lemma 15.1.2 and get a k-independent set covering family $\mathcal{F}'$ for (G',k). Then we output the family

$$\mathcal{F}(G,k) = \{(A \cup B) \setminus N_G(B) \mid A \in \mathcal{F}', B \subseteq S \text{ is an independent set in } G\}.$$

We claim that $\mathcal{F}(G,k)$ is a k-independence covering family for (G,k). Towards that, first we prove that all sets in $\mathcal{F}(G,k)$ are independent sets in G. Let $Y \in \mathcal{F}$. We know that $Y = (A \cup B) \setminus N_G(B)$, for some $A \in \mathcal{F}'$ and $B \subseteq S$ which is an independent set in G. By the definition of $\mathcal{F}'$, A is an independent set in G. Since A and B are independent sets in G, $Y = (A \cup B) \setminus N_G(B)$ is an independent set in G. Now we show that for any independent set X in G of cardinality at most k, there is an independent set containing X in $\mathcal{F}(G,k)$. Let $X = X' \uplus X''$, where $X' = X \setminus S$ and $X'' = X \cap S$. By the definition of $\mathcal{F}'$, there is a set $Z \in \mathcal{F}'$ such that $X' \subseteq Z$. Then the set $(Z \cup X'') \setminus N_G(X'') \in \mathcal{F}(G,k)$ is the required independent set containing X.

Observe that $|\mathcal{F}(G,k)| \leq |\mathcal{F}'| \cdot 2^{|S|}$. Also, the running time of this algorithm is equal to the time taken to compute $\mathcal{F}'$ plus $|\mathcal{F}(G,k)| \cdot (n+m)$. Thus, the running time and the bound on the cardinality of $\mathcal{F}(G,k)$ as claimed in the lemma follows from Lemma 15.1.2 and Remark 15.1.2. □

Remark 15.1.3. *An alternate independence covering family for the situation in Corollary 15.1.1 can be obtained directly from Lemma 15.1.2 by observing that the input graph has degeneracy at most $d + |S|$. This procedure gives an independence covering family whose size (in terms of the dependence on $|S|$) has a factor of $|S|^{\mathcal{O}(k)}$ in contrast to $2^{|S|}$ in Corollary 15.1.1. Thus, the result of Corollary 15.1.1 is relevant only when $d \ll |S| \ll k$.*

15.1.2 Nowhere Dense Graphs

In this section, we show that for any *nowhere dense* graph class [169, 170], there exists a function f such that given an n-vertex graph from this graph class, any real ϵ and any positive integer k, one can construct a k-independence covering family for this graph of size $f(k,\epsilon) \cdot n^{\epsilon}$. The class of graphs that is nowhere dense is a common generalization of proper minor closed classes, classes of graphs with bounded degree, graph class locally excluding a fixed graph H as minor and classes of bounded expansion (see [170, Figure 3]). Also, they are incomparable to the class of bounded degeneracy graphs [172, 36]. In order to define nowhere denseness, we need several new definitions.

Definition 15.1.3 (Shallow minor)**.** *A graph M is an r-*shallow minor *of G, where r is an integer, if there exists a set of disjoint subsets $V_1, \ldots, V_{|M|}$ of $V(G)$ such that*

1. *each graph $G[V_i]$ is connected and has radius at most r, and*
2. *there is a bijection $\psi \colon V(M) \to \{V_1, \ldots, V_{|M|}\}$ such that for every edge $uv \in E(M)$ there is an edge in G with one endpoint in $\psi(u)$ and second in $\psi(v)$.*

The set of all r-shallow minors of a graph G is denoted by $G \triangledown r$. Similarly, the set of all r-shallow minors of all the members of a graph class $\mathcal{G}$ is denoted by $\mathcal{G} \triangledown r = \cup_{G \in \mathcal{G}} (G \triangledown r)$.

We first introduce the definition of a graph class that is nowhere dense; let $\omega(G)$ denotes the size of the largest clique in G and $\omega(\mathcal{G}) = \sup_{G \in \mathcal{G}} \omega(G)$.

Definition 15.1.4 (Nowhere dense)**.** *A graph class $\mathcal{G}$ is* nowhere dense *if there exists a function $f_\omega \colon \mathbb{N} \to \mathbb{N}$ such that for all r we have that $\omega(\mathcal{G} \triangledown r) \leq f_\omega(r)$.*

We refer the readers to the book by Nesetril and Ossona de Mendez [171] for a detailed exposition of nowhere dense classes of graphs, their alternate characterizations and several properties of them. See also [112]. We rely on the following result that bounds the degeneracy of any class of graphs that is nowhere dense, to give a construction for independence covering family for such graph classes.

Proposition 15.1.2 (Corollary 2.6, [112])**.** *Let $\mathcal{G}$ be a class of graphs that is nowhere dense. Then for any real $\epsilon > 0$, for any $G \in \mathcal{G}$, there exists a function f such that the degeneracy of G is $f(\epsilon) \cdot n^\epsilon$.*

We now give the construction of independence covering family for the class of graphs that are nowhere dense.

Lemma 15.1.3. *Let $\mathcal{G}$ be a class of graphs that is nowhere dense. For any $k \in \mathbb{N}$, any $\delta \in \mathbb{R}$ such that $\delta > 0$ and for any $G \in \mathcal{G}$, there is a deterministic algorithm that runs in time $f(\epsilon)^{\mathcal{O}(k)} \cdot n^{1+\delta} \log n$ and outputs a k-independence covering family for (G, k) of size $f(\epsilon)^{\mathcal{O}(k)} \cdot n^\delta \log n$, where f is the function defined in Proposition 15.1.2.*

Proof. Set $\epsilon = \frac{\delta}{2k}$. From Remark 15.1.2, an independence covering family for G of size $2^{\mathcal{O}(k \log d)} \log n$ can be computed in time $2^{\mathcal{O}(k \log d)} n \log n$, where d is the degeneracy of G. Since $G \in \mathcal{G}$ and $\mathcal{G}$ is a class of graphs that are nowhere dense, from Proposition 15.1.2, $d = f(\epsilon) \cdot n^\epsilon$, for some function f. Thus, we obtain an independence covering family for G of size $f(\epsilon)^{\mathcal{O}(k)} \cdot n^\delta \log n$ in time $f(\epsilon)^{\mathcal{O}(k)} \cdot n^{1+\delta} \log n$. □

15.1.3 Barriers

In this subsection we show that we cannot get small independence covering families on general graphs. We also show that we cannot get small covering families when we generalize the notion of "independent set" to something similar even on graphs of bounded degeneracy.

Independence covering family for general graphs. Let k be a positive integer. Consider the graph G on n vertices, where n is divisible by k, which is a disjoint collection of k cliques on $\frac{n}{k}$ vertices each. Let $C_1, \ldots, C_k$ be the disjoint cliques that comprise G. Let $\mathcal{F}(G, k)$ be a k-independence covering family for (G, k). Then, we

claim that, $|\mathcal{F}(G,k)| \geq \left(\frac{n}{k}\right)^k$. Consider the family $\mathcal{I}$ of independent sets of G of size at most k defined as $\mathcal{I} = \{\{v_1, \ldots, v_k\} : \forall i \in [k], v_i \in C_i\}$. Note that $|\mathcal{I}| = \left(\frac{n}{k}\right)^k$. We now prove that it is not the case that there exists $Y \in \mathcal{F}(G,k)$ such that for two distinct sets $X_1, X_2 \in \mathcal{I}$, $X_1, X_2 \subseteq Y$. This would imply that $|\mathcal{F}(G,k)| \geq \left(\frac{n}{k}\right)^k$. Suppose, for the sake of contradiction, that there exists $Y \in \mathcal{F}(G,k)$ and $X_1, X_2 \in \mathcal{I}$ such that $X_1 \neq X_2, X_1 \subseteq Y$ and $X_2 \subseteq Y$. Since $X_1 \neq X_2$, there exist $u \in X_1$ and $v \in X_2$ such that $u, v \in C_i$ for some $i \in [k]$. Since $X_1 \subseteq Y$ and $X_2 \subseteq Y$, $u, v \in Y$, which contradicts the fact that Y is an independent set in G (because $uv \in E(G)$).

Induced matching covering family for disjoint union of stars. We show that if we generalize independent set to induced matching, then we cannot hope for small covering families even on the disjoint union of *star* graphs, which are graphs of degeneracy one.

Definition 15.1.5 (Induced Matching Covering Family)**.** *For a graph G and a positive integer k, a family $\mathcal{M} \subseteq 2^{V(G)}$ is called an* induced matching covering family *for (G,k) if for all $Y \in \mathcal{M}$, $G[Y]$ is a matching, that is, each vertex of Y has degree exactly one in $G[Y]$, and for any induced matching M in G on at most k vertices, there exists $Y \in \mathcal{M}$ such that $V(M) \subseteq Y$.*

Let k be a positive integer. Consider the graph G on n vertices, where $2n$ is divisible by k, which is a disjoint collection of $\frac{k}{2}$ stars on $\frac{2n}{k}$ vertices ($K_{1,\frac{2n}{k}-1}$). That is each connected component of G is isomorphic to $K_{1,\frac{2n}{k}-1}$. Let $\mathcal{R}$ be the set of all maximal matchings in G. Each matching in $\mathcal{R}$ consists of $\frac{k}{2}$ edges, one from each connected component. Observe that all these matchings are induced matchings in G. Union of any two distinct matchings in $\mathcal{R}$ will have a P_3. This implies that the cardinality of any induced matching covering family for (G,k) is at least $|\mathcal{R}| = (\frac{2n}{k} - 1)^{\frac{k}{2}}$.

r-independent covering family for disjoint union of stars. Let G be a graph. For any $r \in \mathbb{N}$, $X \subseteq V(G)$ is called an *r-independent set* in G, if for any $u, v \in V(G)$, $d_G(u,v) > r$. An independent set in G is a 1-independent set in G.

Definition 15.1.6 (r-independent Covering Family)**.** *For any $r \in \mathbb{N}$, for a graph G and a positive integer k, a family $\mathcal{S} \subseteq 2^{V(G)}$ is called a* r-independent covering family *for (G,k) if for all $Y \in \mathcal{S}$, Y is an r-independent set in G and for any $X \subseteq V(G)$ of size at most k such that X is an r-independent set in G, there exists $Y \in \mathcal{S}$ such that $X \subseteq Y$.*

Let k be a positive integer. Consider the graph G on n vertices, where n is divisible by k, which is a disjoint collection of k stars on $\frac{n}{k}$ vertices ($K_{1,\frac{n}{k}-1}$). That is each connected component of G is isomorphic to $K_{1,\frac{n}{k}-1}$. Notice that G is a 1-degenerate graph. Let $C_1, \ldots, C_k$ be the components of G. Define $\mathcal{I} = \{\{v_1, \ldots, v_k\} : \forall i \in [k], v_i \in C_i\}$. Clearly each set in $\mathcal{I}_G()$ is a r-independent set for any $r \in \mathbb{N}$. Moreover, the union of any two distinct sets in $\mathcal{I}_G()$ is not a 2-independent set. This implies that the cardinality of any r-independent covering family for (G, k) is at least $|\mathcal{I}| = \left(\frac{n}{k}\right)^k$ for any $r \geq 2$.

Acyclic covering family for 2-degenerate graphs. We show that covering families for induced acyclic subgraphs on 2-degenerate graphs will have large cardinality.

Definition 15.1.7 (Acyclic Set Covering Family)**.** *For a graph G and a positive integer k, a family $\mathcal{A} \subseteq 2^{V(G)}$ is called an* acyclic set covering family *for (G, k) if for all $Y \in \mathcal{M}$, $G[Y]$ is a forest and for any $X \subseteq V(G)$ of size at most k such that $G[X]$ is a forest, there exists $Y \in \mathcal{A}$ such that $X \subseteq Y$.*

Let k be a positive integer. Consider the graph G on n vertices, where $3n$ is divisible by k, which is a disjoint union of $\frac{k}{3}$ complete bipartite graphs $K_{2,\frac{3n}{k}-2}$. The degeneracy of G is 2. Without loss of generality assume that $\frac{3n}{k}$ is strictly more than 2. Let $H_1, \ldots, H_{\frac{k}{3}}$ be the connected components of G. Let $H_i = (L_i \uplus R_i, E_i)$, where $|L_i| = 2$. Now consider the family of sets $\mathcal{I} = \{L_1 \cup \ldots \cup L_{\frac{k}{3}} \cup \{v_1, \ldots, v_{\frac{k}{3}}\} \mid v_i \in R_i\}$. Each set in $\mathcal{I}_G()$ induces a collection of induced paths on 3 vertices (P_3). Also, the union of any two sets in $\mathcal{I}_G()$ contains a cycle on 4 vertices and hence, not acyclic. This implies that the cardinality of any acyclic set covering family for (G, k) is at least $|\mathcal{I}| = \left(\frac{3n}{k} - 2\right)^{\frac{k}{3}}$.

15.2 Algorithms for Bounded Degeneracy Graphs: Trading Independence for Annotations

In this section we give FPT algorithms for Stable s-t Separator, Stable Odd Cycle Transversal, Stable Multicut and for Stable Directed Feedback Vertex Set on d-degenerate graphs, by applying Lemmas 15.0.1 and 15.1.2. All these algorithms, except the one for Stable Directed Feedback Vertex Set, are later used as a subroutine to design FPT algorithms on general graphs.

We begin by defining a general algorithmic framework that will be applicable to each of the algorithms in this section. To this end, we define Π-Vertex Deletion, Annotated Π-Vertex Deletion and Stable Π-Vertex Deletion problems, for any graph class Π.

Π-Vertex Deletion	**Parameter:** k
Input: An instance $\mathcal{I}$ of a graph problem containing a graph G, an integer k	
Question: Does there exist $S \subseteq V(G)$, such that $\|S\| \leq k$ and $G - S \in \Pi$?	

Annotated Π-Vertex Deletion	**Parameter:** k
Input: An instance $\mathcal{I}$ of a graph problem containing a graph G, a subset $Y \subseteq V(G)$, an integer k	
Question: Does there exist $S \subseteq Y$, such that $\|S\| \leq k$ and $G - S \in \Pi$?	

Stable Π-Vertex Deletion	**Parameter:** k
Input: An instance $\mathcal{I}$ of a graph problem containing a graph G, an integer k	
Question: Does there exist $S \subseteq V(G)$, such that $\|S\| \leq k$, S is an independent set and $G - S \in \Pi$?	

Using our constructions of the independence covering families, the following lemma describes a procedure to design FPT algorithms for Stable Π-Vertex Deletion problems using FPT algorithms for Annotated Π-Vertex Deletion, for graphs of bounded degeneracy. in the following, for any positive integers r, s and $p = r + s$, $N(r, s) = \frac{p\binom{p}{r}}{\log\binom{p}{r}}$.

Lemma 15.2.1. *If there is an algorithm that solves* Annotated Π-Vertex Deletion *on a d-degenerate graph on n vertices in time $T(d, n)$, then* Stable Π-Vertex Deletion *on d-degenerate graphs can be solved by,*

1. *a randomized algorithm with worst case running time $(T(d, n) + (n + m)) \cdot \binom{k(d+1)}{k} \cdot k^2(1+d)$ that always outputs correctly if the instance is a* No *instance and makes an error with probability at most $1 - 1/e$ if it is* Yes *instance; and*
2. *a deterministic algorithm that runs in time $(T(d, n)+(n+m))\cdot N(k, kd)^{1+o(1)} \log n$.*

Proof. We first begin by describing our randomized algorithm[2]. Let $(\mathcal{I}, k)$ be an instance of Stable Π-Vertex Deletion and let G be the graph of the instance

[2]To shave off the log factor in the randomized algorithm, that we would get if we construct

$\mathcal{I}$. Our algorithm runs the following two step procedure $\binom{k(1+d)}{k} \cdot k(1+d)$ many times.

1. Run Algorithm 2 on (G, k) and let Z be its output.
2. Run the algorithm of Annotated Π-Vertex Deletion on the instance $(\mathcal{I}, k, Z)$.

Our algorithm will output Yes, if Step 2 returns Yes at least once. Otherwise, our algorithm will output No. We now prove the correctness of our algorithm. Since in Step 1 the output set Z is always an independent set of G, if the algorithm returns Yes, the input instance is a Yes instance. For the other direction, suppose the input instance is a Yes instance. Let X be a solution to it. Since X is an independent set, from Lemma 15.0.1, $X \subseteq Z$ with probability at least $p = \frac{1}{\binom{k(d+1)}{k} \cdot (k(d+1))}$. Thus, the probability that in all the executions of Step 1, $X \not\subseteq Z$ is at most $(1-p)^{1/p} \leq 1/e$. Therefore, the probability that in at least one execution of Step 1, $X \subseteq Z$, is at least $1 - 1/e$. Now, consider the iteration of the algorithm when $X \subseteq Z$. For this iteration, $(\mathcal{I}, k, Z)$ is a Yes instance of Annotated Π-Vertex Deletion, and thus, our algorithm will output Yes in this iteration. Therefore, if the input instance is a Yes instance, our algorithm will output Yes with probability at least $1 - 1/e$. The running time of our algorithm follows from Lemma 15.0.1 and the running time for Annotated Π-Vertex Deletion.

For our deterministic algorithm, the algorithm first computes a k-independence covering family for (G, k), $\mathcal{F}(G, k)$, using the algorithm of Lemma 15.1.2. For each $Z \in \mathcal{F}(G, k)$, it then solves the instance $(\mathcal{I}, k, Z)$ of Annotated Π-Vertex Deletion. If the algorithm of Annotated Π-Vertex Deletion returns Yes on either of the instances, then our algorithm reports Yes, otherwise it reports No. The correctness of the algorithm follows from the definition of independent set covering family and discussion done in the above paragraph. The running time of the algorithm follows from Lemma 15.1.2 and the running time to solve Annotated Π-Vertex Deletion. □

The rest of the section focuses on four Π-Vertex Deletion problems viz. s-t-Separator, Odd Cycle Transversal (OCT), Directed Feedback Vertex Set (DFVS) and Multicut. In s-t-Separator, the instance $\mathcal{I}$ contains a graph

an independent set covering family using the algorithm of Lemma 15.1.1, we use Algorithm 2 in our algorithm instead of constructing the whole $\mathcal{F}(G, k)$ before hand using multiple rounds of Algorithm 2.

G and $s, t \in V(G)$, and Π is the class of graphs which contain the vertices s, t and, s and t belong to different connected components. In OCT, the instance $\mathcal{I}$ contains a graph G, and Π is the collection of all bipartite graphs. In DFVS, the instance $\mathcal{I}$ contains a directed graph D, and Π is the collection of all acyclic directed graphs. In MULTICUT, the instance $\mathcal{I}$ contains a graph G and a set $T = \{(s_i, t_i) : i \in [p]\}$ of terminal pairs, and Π is the collection of graphs where there is no path from s_i to t_i for each $i \in [p]$. In this chapter, we abuse the notation a little and whenever we refer to the degeneracy of a directed graph, we mean the degeneracy of its underlying undirected graph.

Using the framework of Lemma 15.2.1 and by designing simple algorithms for the ANNOTATED Π-VERTEX DELETION problems corresponding to the above mentioned problems from the algorithms of the corresponding Π-VERTEX DELETION problems, we get the following theorem.

Theorem 15.2.1. *There is a randomized algorithm with one-sided error probability* $1/e$ *and a deterministic algorithm for*

1. STABLE *s-t* SEPARATOR *(*SSTS*) and* STABLE ODD CYCLE TRANSVERSAL *(*SOCT*) on d-degenerate graphs that run in time* $2^{\mathcal{O}(k \log d)} \cdot n$ *and* $2^{\mathcal{O}(k \log d)} \cdot n \log n$ *respectively,*
2. STABLE DIRECTED FEEDBACK VERTEX SET *(*SDFVS*) on d-degenerate graphs that run in time* $(k+1)! \cdot 2^{\mathcal{O}(k \log d)} \cdot n$ *and* $(k+1)! \cdot 2^{\mathcal{O}(k \log d)} \cdot n \log n$ *respectively, and*
3. STABLE MULTICUT *on d-degenerate graphs that run in time* $2^{\mathcal{O}(k^3 + k \log d)} \cdot mn \log^2 n$.[3]

To prove Theorem 15.2.1, it is enough to design appropriate algorithms for the annotated versions of these problems which we do below. Henceforth, an instance of ANNOTATED *s-t*-SEPARATOR (ASTS), ANNOTATED OCT (AOCT), ANNOTATED DFVS (ADFVS) and ANNOTATED MULTICUT is (G, s, t, Y, k), (G, Y, k), (D, Y, k) and (G, T, Y, k) respectively.

Lemma 15.2.2. ASTS *can be solved in time* $\mathcal{O}(k \cdot (n + m))$.

Proof Sketch. To prove the lemma, we apply Proposition 2.4.1 on (G, s, t, w, k), where w is defined as follows: $w(v) = 1$ if $v \in Y$ and $k + 1$ otherwise. □

[3]The randomized algorithm for STABLE MULTICUT does not give any better running time than the deterministic one, so for the sake of soundness of the sentence we may assume that the randomized algorithm is the same as the deterministic algorithm.

We will need the following result about OCT.

Proposition 15.2.1 ([184])**.** OCT *can be solved in time* $\mathcal{O}(4^k \cdot k^4 \cdot (n+m))$.

Using Proposition 15.2.1, we can get the following result about AOCT.

Lemma 15.2.3. AOCT *can be solved in time* $\mathcal{O}(4^k \cdot k^6 \cdot (n+m))$

Proof sketch. We give a polynomial time reduction from AOCT to OCT as follows. We replace each $v \in V(G) \setminus Y$, with $k+1$ vertices $v_1, \ldots v_{k+1}$ with same neighborhood as v, that is, the neighborhood of $v_1, \ldots v_{k+1}$ are same in the resulting graph (see Figure 13.3 for an illustration). Let G' be the resulting graph. Then any minimal odd cycle transversal which contain a vertex from $\{v_1, \ldots, v_{k+1}\}$ will also contain all the vertices in $\{v_1, \ldots, v_{k+1}\}$. Thus to find a k sized solution for AOCT, it is enough to find an odd cycle transversal of size k in G'. The total number of vertices in G' is at most $k|V(G)|$ and the total number of edges in G' is at most $(k+1)^2|E(G)|$. Thus the running time of the algorithm follows from Proposition 15.2.1. □

We need to use the following known algorithm for DFVS/DFVS+Tw-η Mod.

Lemma 15.2.4 ([146])**.** DFVS/DFVS+Tw-η Mod *can be solved in time* $\mathcal{O}((k+1)! \cdot 4^k \cdot k^5 \cdot (n+m))$.

Lemma 15.2.5. ADFVS *can be solved in* $\mathcal{O}((k+1)! \cdot 4^k \cdot k^7 \cdot (n+m))$ *time.*

Proof Sketch. Construct G' as in Lemma 15.2.3, that is, add $k+1$ copies for each vertex in $V(G) \setminus Z$ to the graph G such that all of them have the same neighborhood in the resulting graph. Then apply Lemma 15.2.4 on (G', k). The proof of correctness of this algorithm is similar in arguments to the proof of Lemma 15.2.3. □

Next we state an algorithmic result for Multicut that is used by our algorithm.

Lemma 15.2.6 ([156, 146])**.** Multicut *can be solved in* $2^{\mathcal{O}(k^3)} \cdot mn \log n$ *time.*

Lemma 15.2.7. Annotated Multicut *can be solved in time* $2^{\mathcal{O}(k^3)} \cdot mn \log n$.

Proof sketch. We first give a polynomial time reduction from Annotated Multicut to Multicut which is described below.

Let (G, T, Y, k) be an instance of Annotated Multicut. Construct a graph G' from G by replace each $v \in V(G) \setminus Y$, with $k+1$ vertices $v_1, \ldots v_{k+1}$ with

same neighborhood as v. That is, the neighborhood of $v_1, \dots v_{k+1}$ are same in the resulting graph G'. We call the set of vertices that are added for v in G' as the block for v. We now construct the set of terminal pairs T' from the set of terminals T as follows. If $\{s,t\} \in T$ and $\{s,t\} \subseteq Y$, we add $\{s,t\}$ to T'. Suppose $\{s,t\} \in T$ and $\{s,t\} \cap Y = \{t\}$. Let $s_1, \dots, s_{k+1}$ be the block for s in G'. We add $\{s_1,t\}, \dots \{s_{k+1},t\}$ to T'. Suppose $\{s,t\} \in T$ and $\{s,t\} \subseteq V(G) \setminus Y$. Let $s_1, \dots, s_{k+1}$ and $t_1, \dots, t_{k+1}$ be the blocks for s and t, respectively. We add $\{\{s_i,t_j\} \mid i,j \in [k+1]\}$ to T'.

We will now show that (G,T,Y,k) is a YES instance of ANNOTATED MULTICUT if and only if (G',T',k) is a YES instance of MULTICUT. For the forward direction, let C be a multicut of size at most k in G such that $C \subseteq Y$. We claim that C is a multicut of T' in G'. Suppose not. Then, there is a path from s' to t' in $G' - C$, where $\{s',t'\} \in T'$. Let s and t be the vertices in $V(G)$ such that s' and t' are the vertices corresponding to them, respectively, that is, if $s' \in Y$, then $s = s'$, otherwise let s be the vertex such that s' is in the block of vertices constructed for the replacement of s in G'. By replacing each vertex in the $s' - t'$ path in G' by the corresponding vertex in G, we get a walk from s to t in $G - C$, which contradicts the fact that C is a multicut of T in G. For the backward direction, suppose C' is a minimal multicut of T' in G' of size at most k. Since, for any $v \in V(G) \setminus Y$, the neighborhood of $v_1, \dots v_{k+1}$ in G' is the same as that of v in G and $|C'| \leq k$, $C' \cap \{v_1, \dots, v_{k+1}\} = \emptyset$. Thus, $C' \subseteq Y$. Since G' is a supergraph of G and $T \subseteq T'$, C' is a multicut of T in G.

Thus, to find a k sized multicut of T in G which is fully contained in Y, it is enough to find a multicut of T' in G'. The total number of vertices in G' is at most $k|V(G)|$ and the total number of edges in G' is at most $(k+1)^2|E(G)|$. Thus, the running time of the algorithm follows from Lemma 15.2.6. This completes the proof sketch of the lemma. □

The proof of Theorem 15.2.1 follows from Lemmas 15.2.1, 15.2.2, 15.2.3, 15.2.5 and 15.2.7 and the fact that the number of edges in an n-vertex d-degenerate graph is at most dn.

Chapter 16

Applications of the Independent Set Covering Lemma and Degeneracy Reduction Preserving Minimal Multicuts

Fernau [75] posed as an open problem whether Stable Odd Cycle Transversal is FPT. This problem was resolved by Marx et al. [159], who gave FPT algorithms for Stable s-t Separator running in time $2^{2^{k^{\mathcal{O}(1)}}} \cdot (n+m)$ and Stable Odd Cycle Transversal running in time $2^{2^{k^{\mathcal{O}(1)}}} \cdot (n+m) + \mathcal{O}(3^k \cdot nm)$. Here, the $\mathcal{O}(3^k \cdot nm)$ term in the running time comes from a direct invocation of the algorithm of Reed et al. [187] for Odd Cycle Transversal. Furthermore, Marx et al. [159] gave an algorithm for Stable Multicut with running time $f(k, |T|)(n+m)$ for some function f. They posed as open problems, the problem of determining whether there exists an FPT algorithm for Stable Multicut parameterized by k only, and the problem of determining whether there exists an FPT algorithm for Stable Odd Cycle Transversal with running time $2^{k^{\mathcal{O}(1)}} \cdot (n+m)$. The problem of determining whether there exists an FPT algorithm for Stable Multicut parameterized by k was restated by Michał Pilipczuk at the update meeting on graph separation problems in 2013 [66].

Subsequently, algorithms for Odd Cycle Transversal with running time $4^k k^{\mathcal{O}(1)} \cdot (n+m)$ were found, independently by Iwata et al. [120] and Ramanujan and Saurabh [184]. Replacing the call to the algorithm of Reed et al. [187] in the algorithm of Marx et al. [159] for Stable Odd Cycle Transversal by either of

the two $4^k \cdot k^{\mathcal{O}(1)} \cdot (n+m)$ time algorithms for Odd Cycle Transversal yields a $2^{2^{k^{\mathcal{O}(1)}}} \cdot (n+m)$ time algorithm for Stable Odd Cycle Transversal. However, obtaining a $2^{k^{\mathcal{O}(1)}}(n+m)$ time algorithm still remained an open problem.

Using Lemma 15.1.2, we directly obtained FPT algorithms for Stable s-t Separator, Stable Odd Cycle Transversal, Stable Multicut and Stable Directed Feedback Vertex Set on d-degenerate graphs. At a glance the applicability of Lemma 15.0.1 seems to be limited to problems on graphs of bounded degeneracy. However, there already exist powerful tools in the literature to reduce certain problems on general input graphs to special classes. For us, the *treewidth reduction* of Marx et al. [159] is particularly relevant, since a direct application of their main theorem reduces Stable s-t Separator and Stable Odd Cycle Transversal to the same problems on graphs of bounded treewidth. Since graphs of bounded treewidth have bounded degeneracy, we may now apply our algorithms for bounded degeneracy graphs, obtaining new FPT algorithms for Stable s-t Separator and Stable Odd Cycle Transversal on general graphs. Our algorithms have running time $2^{k^{\mathcal{O}(1)}} \cdot (n+m)$, thus resolving, in the affirmative, one of the open problems of Marx et al. [159].

One of the reasons that the parameterized complexity of Stable Multicut parameterized by the solution size was left open by Marx et al. [159] was that their treewidth reduction does not apply to multi-terminal cut problems when the number of terminals is unbounded. Our degeneracy reduction preserving minimal multicuts from Chapter 13 works for such multi-terminal cut problems.

In this chapter, we resolve all the open problems stated by Marx et al. in [159], viz. the design of single-exponential FPT algorithms for Stable s-t Separator and Stable Odd Cycle Transversal, and the design of an FPT algorithm for Stable Multicut.

16.1 Single-Exponential FPT Algorithms for Stable s-t Separator and Stable Odd Cycle Transversal

For our algoirthms of Stable s-t Separator and Stable Odd Cycle Transversal on general graphs, the core is the Treewidth Reduction Theorem of [159] and our algorithms for SSTS and SOCT on bounded degeneracy graphs from Theo-

rem 15.2.1. We begin by stating the Treewidth Reduction Theorem.

Theorem 16.1.1 (Treewidth Reduction Theorem, Theorem 2.15 [159]). *Let G be a graph, $T \subseteq V(G)$ and $k \in \mathbb{N}$. Let C be the set of all vertices of G participating in a minimal s-t-separator of cardinality at most k for some $s, t \in T$. For every k and $|T|$, there is an algorithm that computes a graph $G^\star$ having the following properties, in time $2^{(k+|T|)^{\mathcal{O}(1)}} \cdot (n+m)$.*

1. *$C \cup T \subseteq V(G^\star)$,*
2. *for every $s, t \in T$, a set $K \subseteq V(G^\star)$ with $|K| \leq k$ is a minimal s-t-separator of $G^\star$ if and only if $K \subseteq C \cup T$ and K is a minimal s-t-separator of G,*
3. *the treewidth of $G^\star$ is at most $2^{(k+|T|)^{\mathcal{O}(1)}}$, and*
4. *$G^\star[C \cup T]$ is isomorphic to $G[C \cup T]$.*

We remark here that Theorem 2.15 in [159] does not state the explicit dependence on k and $|T|$ in the running time of the algorithm and the treewidth of $G^\star$.

Stable s-t Separator. Let (G, k) be an instance of SSTS. To solve SSTS on general graphs, we first apply the Treewidth Reduction Theorem (Theorem 16.1.1) on $G, T = \{s, t\}$ and k to obtain a graph $G^\star$ with treewidth upper bounded by $2^{k^{\mathcal{O}(1)}}$. From [159], to solve SSTS, it is enough to work with this new graph $G^\star$. By conditions 2 and 4, to find a minimal independent s-t-separator separator in G, it is enough to find a minimal independent s-t-separator in $G^\star$. Since treewidth of any graph is at most its degeneracy, we have that the degeneracy of $G^\star$ is at most $2^{k^{\mathcal{O}(1)}}$, and hence we apply Theorem 15.2.1 to get a solution of SSTS on (G, k). That is, we get the following theorem.

Theorem 16.1.2. *There is a randomized algorithm that solves* SSTS *in time $2^{k^{\mathcal{O}(1)}}(n+m)$ with success probability at least $1 - \frac{1}{e}$. There is a deterministic algorithm that solves* SSTS *in time $2^{k^{\mathcal{O}(1)}}(n+m)\log n$.*

Stable Odd Cycle Transversal. By using Theorem 16.1.2 and Proposition 15.2.1 we get a $2^{k^{\mathcal{O}(1)}}(n+m)$ time (FPT linear time) algorithm for SOCT. Towards that, in Theorem 4.2 of Marx et al. [159] we replace the algorithm of Kawarabayashi and Reed [128] with Proposition 15.2.1 and the algorithm for SSTS with Theorem 16.1.2. For completeness we include the proof here.

Proposition 16.1.1 (Lemma 4.1, [159]). *Let G be a bipartite graph and let (B', W') be a proper 2-coloring of the vertices. Let B and W be two subsets of $V(G)$. Then, for any $S \subseteq V(G)$, the graph $G - S$ has a 2-coloring where $B \setminus S$ is black and $W \setminus S$ is white if and only if S separates $X := (B \cap B') \cup (W \cap W')$ and $Y := (B \cap W') \cup (W \cap B')$.*

Theorem 16.1.3. *There is a randomized algorithm that solves* SOCT *in time $2^{k^{\mathcal{O}(1)}}(n+m)$ with success probability at least $1 - \frac{1}{e}$. There is a deterministic algorithm that solves* OCT *in time $2^{k^{\mathcal{O}(1)}}(n+m)\log n$.*

Proof. Using the algorithm of Proposition 15.2.1, find a set $S_0 \subseteq V(G)$ of size at most k such that $G \setminus S_0$ is a bipartite graph. Observe that if such a set does not exist, then (G, k) is a NO instance of SOCT. Henceforth, we can assume that such a set S_0 exists. Next, we branch into $3^{|S_0|}$ cases, where each branch has the following interpretation. If we fix a hypothetical solution S and a proper 2-coloring of $G - S$, then each vertex of S_0 is either removed (that is, belongs to S), colored with the first color, say black, or colored with the second color, say white. For a particular branch, let R be the vertices of S_0 to be removed (in order to get the hypothetical solution S) and let B_0 (respectively W_0) be the vertices of S_0 getting color black (respectively white) in a proper 2-coloring of $G - S$. A set S is said to be compatible with the partition (R, B_0, W_0), if $S \cap S_0 = R$ and $G \setminus S$ has a proper 2-coloring, with colors black and white, where the vertices in B_0 are colored black and the vertices in W_0 are colored white. Observe that (G, k) is a YES instance of SOCT if and only if for at least one branch corresponding to a partition (R, B_0, W_0) of S_0, there is a set S compatible with (R, B_0, W_0) of size at most k and S is an independent set. Note that we need to check only those branches corresponding to the partition (R, B_0, W_0) where $G[B_0]$ and $G[W_0]$ are edgeless graphs.

The next step is to transform the problem of finding a set compatible with (R, B_0, W_0) into a separation problem. Let (B', W') be a 2-coloring of $G - S_0$. Let $B = N(W_0) \setminus S_0$ and $W = N(B_0) \setminus S_0$. Let X and Y be the sets as defined in Proposition 16.1.1. That is, $X = (B \cap B') \cup (W \cap W')$ and $Y = (B \cap W') \cup (W \cap B')$. Construct a graph G' that is obtained from G by deleting the set $B_0 \cup W_0$, adding a new vertex s adjacent with $X \cup R$ and adding a new vertex t adjacent with $Y \cup R$. Notice that every s-t-separator in G' contains R. By Proposition 16.1.1, a set S is compatible with (R, B_0, W_0) if and only if S is an $s - t$ separator in G. Thus, we need to decide whether there is an s-t-separator S of size at most k such that $G'[S] = G[S]$ is an edgeless graph and this step can be done by Theorem 16.1.2.

Towards the run time analysis, we run the algorithm of Proposition 15.2.1 once, which takes time $2^{\mathcal{O}(k)}(m+n)$. Then we apply Theorem 16.1.2 at most 3^k times. Thus, we get the required running time. □

16.2 Stable Multicut is FPT

Using our algorithm of Theorem 15.2.1 for Stable Multicut on bounded degeneracy graphs and the Degeneracy Reduction Lemma (Lemma 13.3.1), we are now ready we prove that Stable Multicut is FPT. Theorem 13.0.1 reduces the Stable Multicut problem on general graphs to graphs excluding a clique of size $2^{\mathcal{O}(k)}$ as a topological minor. Since such graphs have bounded degeneracy [32, 134], our algorithm for Stable Multicut on graphs of bounded degeneracy yields an FPT algorithm for the problem on general graphs. ,

The following lemma establishes a relationship between the degeneracy of the graph and the k-connected sets in the graph.

Lemma 16.2.1. *Let $k, d \in \mathbb{N}$ be such that $k \leq d+1$. Let G be a graph which does not contain a k-connected set of size at least d. Then the degeneracy of G is at most $4d-1$.*

Proof. For the sake of contradiction, assume that the degeneracy of G is at least $4d$. Then, by Lemma 13.4.3, there is a $(d+1)$-connected subgraph H of G. Since $k \leq d+1$ and $|V(H)| \geq d+2$, we have that $V(H)$ is a k-connected set in G of size at least $d+2$, which is a contradiction. □

Theorem 16.2.1. Stable Multicut *can be solved in time $2^{\mathcal{O}(k^3)} \cdot n^6 \log n$.*

Proof. Let (G, k) be an instance of Stable Multicut. First, we apply Lemma 13.3.1 and get an equivalent instance $(G^\star, T^\star)$, where $G^\star$ does not contain any $(k+2)$-connected set of size $64^{k+2} \cdot 4(k+2)^2$. Then, by Lemma 16.2.1, the degeneracy of $G^\star$ is at most $64^{k+2} \cdot 16(k+2)^2 - 1$. Then, using Theorem 15.2.1 we get the solution. The running time of the algorithm follows from Lemma 13.3.1 and Theorem 15.2.1. □

Bibliography

[1] Open Problems in Parameterized Complexity. `http://fpt.wikidot.com/open-problems`.

[2] Faisal N Abu-Khzam and Henning Fernau. Kernels: Annotated, proper and induced. In *International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 264–275. Springer, 2006.

[3] Akanksha Agrawal, Daniel Lokshtanov, Amer E. Mouawad, and Saket Saurabh. Simultaneous Feedback Vertex Set: A Parameterized Perspective. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 47, pages 7:1–7:15, 2016.

[4] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):1–27, 2008.

[5] Noga Alon. Ranking tournaments. *SIAM Journal on Discrete Mathematics (SIDMA)*, 20(1):137–142, 2006.

[6] Noga Alon, Béla Bollobás, Michael Krivelevich, and Benny Sudakov. Maximum cuts and judicious partitions in graphs without short cycles. *Journal of Combinatorial Theory, Series B (JCTB)*, 88(2):329–346, 2003.

[7] Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 49–58, 2009.

[8] Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $(1+\varepsilon)$-approximate flow sparsifiers. In *ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 279–293, 2014.

[9] Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics (SIDMA)*, 12(3):289–297, 1999.

[10] Jørgen Bang-Jensen, Alessandro Maddaloni, and Saket Saurabh. Algorithms and kernels for feedback set problems in generalizations of tournaments. *Algorithmica*, 76(2):320–343, 2016.

[11] Jørgen Bang-Jensen and Caresten Thomassen. A polynomial algorithm for the 2-path problem for semicomplete digraphs. *SIAM Journal on Discrete Mathematics (SIDMA)*, 5(3):366–376, 1992.

[12] Reuven Bar-Yehuda, Dan Geiger, Joseph Naor, and Ron M Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. *SIAM Journal on Computing (SICOMP)*, 27(4):942–959, 1998.

[13] Florian Barbero, Gregory Gutin, Mark Jones, and Bin Sheng. Parameterized and approximation algorithms for the load coloring problem. *Algorithmica*, pages 1–19, 2015.

[14] Florian Barbero, Christophe Paul, and Michał Pilipczuk. Exploring the complexity of layout parameters in tournaments and semi-complete digraphs. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 80, pages 70:1–70:13, 2017.

[15] Surender Baswana, Keerti Choudhary, Moazzam Hussain, and Liam Roditty. Approximate Single-Source Fault Tolerant Shortest Path. *ACM Transactions on Algorithms (TALG)*, 16(4):1–22, 2020.

[16] Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant reachability for directed graphs. In *International Symposium on Distributed Computing*, pages 528–543. Springer, 2015.

[17] Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault-tolerant subgraph for single-source reachability: General and optimal. *SIAM Journal on Computing (SICOMP)*, 47(1):80–95, 2018.

[18] Ann Becker and Dan Geiger. Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83(1):167–188, 1996.

[19] Benjamin Bergougnoux, Eduard Eiben, Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Towards a polynomial kernel for directed feedback vertex set. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 83, pages 36:1–36:15. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[20] Stéphane Bessy, Fedor V. Fomin, Serge Gaspers, Christophe Paul, Anthony Perez, Saket Saurabh, and Stéphan Thomassé. Kernels for feedback arc set in tournaments. *Journal of Computer and System Sciences (JCSS)*, 77(6):1071–1078, 2011.

[21] Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. In *European Symposium on Algorithms (ESA)*, pages 167–178. Springer, 2015.

[22] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. In *European Symposium on Algorithms (ESA)*, pages 137–148. Springer, 2014.

[23] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing (SICOMP)*, 25(6):1305–1317, 1996.

[24] Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. *Journal pf the ACM (JACM)*, 63(5):44:1–44:69, 2016.

[25] Béla Bollobás and Alex D. Scott. Judicious partitions of graphs. *Periodica Mathematica Hungarica*, 26(2):125–137, 1993.

[26] Béla Bollobás and Alex D. Scott. Judicious partitions of hypergraphs. *Journal of Combinatorial Theory, Series A (JCTA)*, 78(1):15–31, 1997.

[27] Béla Bollobás and Alex D. Scott. Exact bounds for judicious partitions of graphs. *Combinatorica*, 19(4):473–486, 1999.

[28] Béla Bollobás and Alex D. Scott. Judicious partitions of 3-uniform hypergraphs. *European Journal of Combinatorics*, 21(3):289–300, 2000.

[29] Béla Bollobás and Alex D. Scott. Problems and results on judicious partitions. *Random Structures & Algorithms*, 21(3-4):414–430, 2002.

[30] Béla Bollobás and Alex D. Scott. Judicious partitions of bounded-degree graphs. *Journal of Graph Theory*, 46(2):131–143, 2004.

[31] Béla Bollobás and Alex D. Scott. Max k-cut and judicious k-partitions. *Discrete Mathematics*, 310(15):2126–2139, 2010.

[32] Béla Bollobás and Andrew Thomason. Proof of a conjecture of Mader, Erdös and Hajnal on topological complete subgraphs. *European Journal of Combinatorics*, 19(8):883–887, 1998.

[33] Marthe Bonamy, Łukasz Kowalik, Jesper Nederlof, Michał Pilipczuk, Arkadiusz Socała, and Marcin Wrochna. On directed feedback vertex set parameterized by treewidth. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 65–78. Springer, 2018.

[34] Edouard Bonnet, Bruno Escoffier, Vangelis Th. Paschos, and Emeric Tourniaire. Multi-parameter analysis for local graph partitioning problems: Using greediness for parameterization. *Algorithmica*, 71(3):566–580, 2015.

[35] Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. *SIAM Journal on Computing (SICOMP)*, 47(1):166–207, 2018.

[36] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999.

[37] Nader H. Bshouty and Ariel Gabizon. Almost optimal cover-free families. In *International Conference on Algorithms and Complexity (CIAC)*, pages 140–151. Springer, 2017.

[38] Leizhen Cai. Parameterized complexity of cardinality constrained optimization problems. *The Computer Journal*, 51(1):102–121, 2008.

[39] Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, 73(1):63–86, 2015.

[40] Diptarka Chakraborty and Debarati Das. Sparse weight tolerant subgraph for single source shortest path. In *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 15:1–15:15. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[41] Pierre Charbit, Stéphan Thomassé, and Anders Yeo. The minimum feedback arc set problem is NP-hard for tournaments. *Combinatorics, Probability & Computing*, 16(1):1–4, 2007.

[42] Moses Charikar, Tom Leighton, Shi Li, and Ankur Moitra. Vertex sparsifiers and abstract rounding algorithms. In *Symposium on Foundations of Computer Science (FOCS)*, pages 265–274. IEEE, 2010.

[43] Shiri Chechik. Fault-tolerant compact routing schemes for general graphs. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 101–112. Springer, 2011.

[44] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault tolerant spanners for general graphs. *SIAM Journal on Computing (SICOMP)*, 39(7):3403–3423, 2010.

[45] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. F-sensitivity distance oracles and routing schemes. *Algorithmica*, 63(4):861–882, 2012.

[46] Chandra Chekuri and Vivek Madan. Constant factor approximation for subset feedback set problems via a new LP relaxation. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 808–820, 2016.

[47] Chandra Chekuri and Vivek Madan. Simple and fast rounding algorithms for directed and node-weighted multiway cut. In *ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 797–807. SIAM, 2016.

[48] Jianer Chen, Fedor V Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences (JCSS)*, 74(7):1188–1198, 2008.

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

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

[51] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM (JACM)*, 55(5):21, 2008.

[52] Zhibin Chen, Jie Ma, and Wenan Zang. Coloring digraphs with forbidden cycles. *Journal of Combinatorial Theory, Series B*, 115:210–223, 2015.

[53] Rajesh Chitnis. *Directed Graphs: Fixed-Parameter Tractability & Beyond.* PhD thesis, 2014.

[54] Rajesh Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Transactions on Algorithms (TALG)*, 11(4):28, 2015.

[55] Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michał Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM Journal on Computing (SICOMP)*, 45(4):1171–1229, 2016.

[56] Rajesh Chitnis and Mohammad Taghi Hajiaghayi. Shadowless solutions for fixed-parameter tractability of directed graphs. *Encyclopedia of Algorithms*, pages 1–5, 2008.

[57] Rajesh Chitnis, Mohammad Taghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. *SIAM Journal on Computing (SICOMP)*, 42(4):1674–1696, 2013.

[58] Hyeong-Ah Choi, Kazuo Nakajima, and Chong S. Rim. Graph bipartization and via minimization. *SIAM Journal on Discrete Mathematics (SIDMA)*, 2(1):38–47, 1989.

[59] Maria Chudnovsky, Alexandra Ovetsky Fradkin, and Paul D. Seymour. Tournament immersion and cutwidth. *Journal of Combinatorial Theory, Series B (JCT B)*, 102(1):93–101, 2012.

[60] Maria Chudnovsky and Paul D. Seymour. A well-quasi-order for tournaments. *Journal of Combinatorial Theory, Series B (JCT B)*, 101(1):47–53, 2011.

[61] Julia Chuzhoy. On vertex sparsifiers with steiner nodes. In *ACM Symposium on Theory of Computing (STOC)*, pages 673–688, 2012.

[62] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms.* MIT press, 2009.

[63] Robert Crowston, Gregory Gutin, Mark Jones, and Gabriele Muciaccia. Maximum balanced subgraph problem parameterized above lower bound. *Theoretical Computer Science*, 513:53–64, 2013.

[64] Robert Crowston, Mark Jones, and Matthias Mnich. Max-cut parameterized above the Edwards-Erdős bound. *Algorithmica*, 72(3):734–757, 2015.

[65] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[66] Marek Cygan, Łukasz Kowalik, and Marcin Pilipczuk. Open problems from the update meeting on graph separation problems. http://worker2013.mimuw.edu.pl/slides/update-opl.pdf, 2013.

[67] Marek Cygan, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Clique cover and graph separation: New incompressibility results. *ACM Transactions on Computation Theory (TOCT)*, 6(2):1–19, 2014.

[68] Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory of Computing Systems*, 54(1):73–82, 2014.

[69] Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Minimum bisection is fixed-parameter tractable. *SIAM Journal on Computing (SICOMP)*, 48(2):417–450, 2019.

[70] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Joham M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Symposium on Foundations of Computer Science (FOCS)*, pages 150–159. IEEE, 2011.

[71] Marek Cygan, Marcin Pilipczuk, and Michał Pilipczuk. On group feedback vertex set parameterized by the size of the cutset. *Algorithmica*, 74(2):630–642, 2016.

[72] Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM Journal on Discrete Mathematics (SIDMA)*, 27(1):290–309, 2013.

[73] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing (SICOMP)*, 23:864–894, 1994.

[74] Holger Dell and Dieter Van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *Journal of the ACM (JACM)*, 61(4):23, 2014.

[75] Erik Demaine, Gregory Gutin, Dániel Marx, and Ulrike Stege. 07281 Open problems–structure theory and FPT algorithmcs for graphs, digraphs and hypergraphs. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

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

[77] Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 169–178, 2011.

[78] Irit Dinur and Shmuel Safra. The importance of being biased. In *ACM Symposium on Theory of Computing (STOC)*, pages 33–42, 2002.

[79] Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *Journal of Discrete Algorithms*, 8(1):76–86, 2010.

[80] Rodney G. Downey and Michael R. Fellows. Fixed-parameter intractability. In *Structure in Complexity Theory Conference*, pages 36–37. IEEE Computer Society, 1992.

[81] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing (SICOMP)*, 24(4):873–921, 1995.

[82] Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[83] Rodney G. Downey and Michael R. Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

[84] Christopher S. Edwards. Some extremal properties of bipartite subgraphs. *Canadian Journal of Mathematics*, 25(3):475–483, 1973.

[85] Christopher S. Edwards. An improved lower bound for the number of edges in a largest bipartite subgraph. In *Czechoslovak Symposium on Graph Theory, Prague*, pages 167–181, 1975.

[86] Alina Ene, Jan Vondrák, and Yi Wu. Local distribution and the symmetry gap: Approximability of multiway partitioning problems. In *ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 306–325, 2013.

[87] Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Racke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM Journal on Computing (SICOMP)*, 43(4):1239–1262, 2014.

[88] P Erdős and L Pósa. On independent circuits contained in a graph. *Canadian Journal of Mathematics*, 17:347–352, 1965.

[89] Michael Etscheid and Matthias Mnich. Linear kernels and linear-time algorithms for finding large cuts. In *International Symposium on Algorithms and Computation, (ISAAC)*, pages 31:1–31:13, 2016.

[90] Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

[91] Uriel Feige. Faster FAST (feedback arc set in tournaments). *arXiv preprint arXiv:0911.5094*, 2009.

[92] Paola Festa, Panos M. Pardalos, and Mauricio G. C. Resende. Feedback set problems. In *Handbook of Combinatorial Optimization*, pages 209–258. Springer, 1999.

[93] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[94] Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. *Journal of the ACM (JACM)*, 66(2):1–23, 2019.

[95] Fedor V. Fomin, Serge Gaspers, Artem V. Pyatkin, and Igor Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008.

[96] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A Measure & Conquer approach for the analysis of exact algorithms. *Journal of the ACM (JACM)*, 56(5), 2009.

[97] Fedor V. Fomin, Pinar Heggernes, Dieter Kratsch, Charis Papadopoulos, and Yngve Villanger. Enumerating minimal subset feedback vertex sets. *Algorithmica*, 69(1):216–231, 2014.

[98] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer, 2010. An EATCS Series: Texts in Theoretical Computer Science.

[99] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation, kernelization and optimal FPT algorithms. In *Symposium on Foundations of Computer Science (FOCS)*, pages 470–479. IEEE, 2012. See http://www.ii.uib.no/ daniello/papers/PFDFullV1.pdf for the full version.

[100] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of parameterized preprocessing*. Cambridge University Press, 2018.

[101] Fedor V Fomin and Michał Pilipczuk. Jungles, bundles, and fixed-parameter tractability. In *ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 396–413, 2013.

[102] Fedor V. Fomin and Michał Pilipczuk. Subexponential parameterized algorithm for computing the cutwidth of a semi-complete digraph. In *European Symposium on Algorithms (ESA)*, pages 505–516. Springer, 2013.

[103] Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015.

[104] Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[105] Alexandra O. Fradkin and Paul D. Seymour. Tournament pathwidth and topological containment. *Journal of Combinatorial Theory, Series B (JCTB)*, 103(3):374–384, 2013.

[106] Alexandra O. Fradkin and Paul D. Seymour. Edge-disjoint paths in digraphs with bounded independence number. *Journal of Combinatorial Theory, Series B (JCTB)*, 110:19–46, 2015.

[107] Alexandra O. Fradkin and Paul D. Seymour. Edge-disjoint paths in digraphs with bounded independence number. *Journal of Combinatorial Theory, Series B (JCTB)*, 110:19–46, 2015.

[108] Tibor Gallai. On directed paths and circuits. *Theory of graphs*, 38:2054, 1968.

[109] Tibor Gallai and Arthur Norton Milgram. Verallgemeinerung eines graphentheoretischen satzes von rédei: Ladislaus rédei zum 60. geburtstag. *Acta scientiarum mathematicarum*, 21(3-4):181–186, 1960.

[110] Georges Gardarin and Stefano Spaccapietra. Integrity of databases: A general lockout algorithm with deadlock avoidance. In *IFIP Working Conference on Modelling in Data Base Management Systems*, pages 395–412, 1976.

[111] Geoffrey R Grimmett. An upper bound for the number of spanning trees of a graph. *Discrete Mathematics*, 16(4):323–324, 1976.

[112] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Characterisations of nowhere dense graphs (Invited Talk). In *IARCS Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[113] Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discrete Optimization*, 8(1):61–71, 2011.

[114] Venkatesan Guruswami and Euiwoong Lee. Inapproximability of H-transversal/packing. *SIAM Journal on Discrete Mathematics (SIDMA)*, 31(3):1552–1571, 2017.

[115] Gregory Gutin and Mark Jones. Parameterized algorithms for load coloring problem. *Information Processing Letters*, 114(8):446–449, 2014.

[116] Gregory Gutin and Anders Yeo. Note on maximal bisection above tight lower bound. *Information Processing Letters*, 110(21):966–969, 2010.

[117] Gregory Gutin and Anders Yeo. Constraint satisfaction problems parameterized above or below tight bounds: A survey. In *The Multivariate Algorithmic Revolution and Beyond*, pages 257–286. Springer, 2012.

[118] Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a graph. In *ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 165–174, 1992.

[119] John Haslegrave. Judicious partitions of uniform hypergraphs. *Combinatorica*, 34(5):561–572, 2014.

[120] Yoichi Iwata, Keigo Oka, and Yuichi Yoshida. Linear-time FPT algorithms via network flow. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1749–1761, 2014.

[121] Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, lp-branching, and fpt algorithms. *SIAM Journal on Computing (SICOMP)*, 45(4):1377–1411, 2016.

[122] Naonori Kakimura, Ken-ichi Kawarabayashi, and Yusuke Kobayashi. Erdős-pósa property and its algorithmic applications—parity constraints, subset feedback set, and subset packing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1726–1736, 2012.

[123] Naonori Kakimura, Ken-ichi Kawarabayashi, and Dániel Marx. Packing cycles through prescribed vertices. *Journal of Combinatorial Theory, Series B (JCT B)*, 101(5):378–381, 2011.

[124] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[125] Marek Karpinski and Warren Schudy. Faster algorithms for feedback arc set tournament, Kemeny rank aggregation and betweenness tournament. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 3–14. Springer, 2010.

[126] Ken-ichi Kawarabayashi and Yusuke Kobayashi. Fixed-parameter tractability for the subset feedback set problem and the S-cycle packing problem. *Journal of Combinatorial Theory, Series B (JCT B)*, 102(4):1020–1034, 2012.

[127] Ken-ichi Kawarabayashi, Daniel Král', Marek Krcál, and Stephan Kreutzer. Packing directed cycles through a specified vertex set. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 365–377, 2013.

[128] Ken-ichi Kawarabayashi and Bruce A. Reed. An (almost) linear time algorithm for odd cycles transversal. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 365–378, 2010.

[129] Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *Symposium on Foundations of Computer Science*, pages 160–169. IEEE, 2011.

[130] Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *ACM Symposium on Theory of Computing (STOC)*, pages 95–103, 2007.

[131] Subhash A Khot and Nisheeth K Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative-type metrics into ℓ_1. *Journal of the ACM (JACM)*, 62(1):8, 2015.

[132] Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014.

[133] Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014.

[134] János Komlós and Endre Szemerédi. Topological cliques in graphs II. *Combinatorics, Probability & Computing*, 5:79–90, 1996.

[135] Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Fixed-parameter tractability of multicut in directed acyclic graphs. *SIAM Journal on Discrete Mathematics (SIDMA)*, 29(1):122–144, 2015.

[136] Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *Journal of the ACM (JACM)*, 67(3):16:1–16:50, 2020.

[137] Mithilesh Kumar and Daniel Lokshtanov. Faster exact and parameterized algorithm for feedback vertex set in tournaments. In *Symposium on Theoretical Aspects of Computer Science, (STACS)*, volume 47, pages 49:1–49:13, 2016.

[138] Choongbum Lee, Po-Shen Loh, and Benny Sudakov. Judicious partitions of directed graphs. *Random Structures & Algorithms*, 48(1):147–170, 2016.

[139] Frank Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *ACM Symposium on Theory of Computing (STOC)*, pages 47–56, 2010.

[140] Shaohua Li and Marcin Pilipczuk. An improved FPT algorithm for independent feedback vertex set. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 344–355. Springer, 2018.

[141] Daniel Lokshtanov and Dániel Marx. Clustering with local restrictions. *Information and Computation*, 222:278–292, 2013.

[142] Daniel Lokshtanov, Pranabendu Misra, Joydeep Mukherjee, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. A 2-approximating feedback vertex set in tournaments. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1010–1018, 2020.

[143] Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms (TALG)*, 11(2):1–31, 2014.

[144] Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms (TALG)*, 11(2):15:1–15:31, 2014.

[145] Daniel Lokshtanov and M. S. Ramanujan. Parameterized tractability of multiway cut with parity constraints. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 750–761. Springer, 2012.

[146] Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. A linear time parameterized algorithm for directed feedback vertex set. *arXiv preprint arXiv:1609.04347*, 2016.

[147] Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. A linear time parameterized algorithm for node unique label cover. *arXiv preprint arXiv:1604.08764*, 2016.

[148] Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. Linear time parameterized algorithms for subset feedback vertex set. *ACM Transactions on Algorithms (TALG)*, 14(1):1–37, 2018.

[149] Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. When recursion is better than iteration: A linear-time algorithm for acyclicity with few error vertices. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1916–1933, 2018.

[150] Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2181–2200, 2020.

[151] Jie Ma and Xingxing Yu. On judicious bipartitions of graphs. *Combinatorica*, 36(5):537–556, 2016.

[152] W. Mader. Existenzn-fach zusammenhängender teilgraphen in graphen genügend großer kantendichte. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 37(1):86–97, 1972.

[153] Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *Journal of Algorithms*, 31(2):335–354, 1999.

[154] Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. Parameterizing above or below guaranteed values. *Journal of Computer and System Sciences (JCSS)*, 75(2):137–153, 2009.

[155] Dániel Marx. Some open problems in parameterized complexity. `http://www.cs.bme.hu/dmarx/papers/marx-dagstuhl2017-open.pdf`, 2-17.

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

[157] Dániel Marx. Important separators and parameterized algorithms. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 5–10. Springer, 2011.

[158] Dániel Marx. What's next? future directions in parameterized complexity. In *The Multivariate Algorithmic Revolution and Beyond*, pages 469–496. Springer, 2012.

[159] Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms (TALG)*, 9(4):30, 2013.

[160] Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM Journal on Computing (SICOMP)*, 43(2):355–388, 2014.

[161] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983.

[162] Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theoretical Computer Science*, 461:65–75, 2012.

[163] Pranabendu Misra, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. A polynomial kernel for feedback arc set on bipartite tournaments. *Theory of Computing Systems*, 53(4):609–620, 2013.

[164] Matthias Mnich and Erik Jan van Leeuwen. Polynomial kernels for deletion to classes of acyclic digraphs. *Discrete Optimization*, 25:48–76, 2017.

[165] Matthias Mnich, Virginia Vassilevska Williams, and László A. Végh. A 7/3-approximation for feedback vertex sets in tournaments. In *European Symposium on Algorithms (ESA)*, pages 67:1–67:14, 2016.

[166] Matthias Mnich and Rico Zenklusen. Bisections above tight lower bounds. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 184–193. Springer, 2012.

[167] John W. Moon and Leo Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, 1965.

[168] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *Symposium on Foundations of Computer Science (FOCS)*, pages 182–191. IEEE, 1995.

[169] Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008.

[170] Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011.

[171] Jaroslav Nešetřil and Patrice Ossona De Mendez. *Sparsity: graphs, structures, and algorithms*, volume 28. Springer Science & Business Media, 2012.

[172] Jaroslav Nešetril and Patrice Ossona de Mendez. From sparse graphs to nowhere dense structures: Decompositions, independence, dualities and limits. In *European Congress of Mathematics*, pages 135–165, 2009.

[173] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[174] Merav Parter. Dual failure resilient BFS structure. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 481–490, 2015.

[175] Merav Parter. Fault-tolerant logical network structures. *Bulletin of EATCS*, 1(118), 2016.

[176] Merav Parter. Vertex fault tolerant additive spanners. *Distributed Computing*, 30(5):357–372, 2017.

[177] Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *European Symposium on Algorithms (ESA)*, pages 779–790. Springer, 2013.

[178] Marcin Pilipczuk and Magnus Wahlström. Directed multicut is W[1]-hard, even for four terminal pairs. *ACM Transactions on Computation Theory (TOCT)*, 10(3):13, 2018.

[179] Michał Pilipczuk. Computing cutwidth and pathwidth of semi-complete digraphs via degree orderings. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 20, pages 197–208, 2013.

[180] Matteo Pontecorvi and Paul Wollan. Disjoint cycles intersecting a set of vertices. *Journal of Combinatorial Theory, Series B (JCTB)*, 102(5):1134–1141, 2012.

[181] Venkatesh Raman and Saket Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458, 2006.

[182] Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Transactions on Algorithms (TALG)*, 2(3):403–415, 2006.

[183] Venkatesh Raman, Saket Saurabh, and Ondrej Suchý. An FPT algorithm for tree deletion set. *Journal of Graph Algorithms and Applications*, 17(6):615–628, 2013.

[184] M. S. Ramanujan and Saket Saurabh. Linear-time parameterized algorithms via skew-symmetric multicuts. *ACM Transactions on Algorithms (TALG)*, 13(4):1–25, 2017.

[185] Igor Razgon. Exact computation of maximum induced forest. In *candinavian Workshop on Algorithm Theory (SWAT)*, volume 4059, pages 160–171, 2006.

[186] Bruce Reed, Neil Robertson, Paul D. Seymour, and Robin Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996.

[187] Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

[188] Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B (JCTB)*, 63(1):65–110, 1995.

[189] Bernard Roy. Nombre chromatique et plus longs chemins d'un graphe. *Revue française d'informatique et de recherche opérationnelle*, 1(5):129–132, 1967.

[190] Saket Saurabh and Meirav Zehavi. (k,n-k)-Max-Cut: An $\mathcal{O}^*(2^p)$-time algorithm and a polynomial kernel. *Algorithmica*, 80(12):3844–3860, 2018.

[191] Alex D. Scott. Judicious partitions and related problems. *Surveys in Combinatorics*, 327:95–117, 2005.

[192] Paul D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.

[193] Paul D. Seymour. Packing circuits in eulerian digraphs. *Combinatorica*, 16(2):223–231, 1996.

[194] Hadas Shachnai and Meirav Zehavi. Representative families: A unified tradeoff-based approach. *Journal of Computer and System Sciences (JCSS)*, 82(3):488–502, 2016.

[195] Abraham Silberschatz, Greg Gagne, and Peter B Galvin. *Operating System Concepts (9th edition)*. Wiley, 2012.

[196] Yinglei Song. An improved parameterized algorithm for the independent feedback vertex set problem. *Theoretical Computer Science*, 535:25–30, 2014.

[197] Ewald Speckenmeyer. On feedback problems in digraphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 218–231. Springer, 1989.

[198] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.

[199] Benny Sudakov. Graph theory. *Lecture Notes*, 2016.

[200] Yuma Tamura, Takehiro Ito, and Xiao Zhou. Algorithms for the independent feedback vertex set problem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 98(6):1179–1188, 2015.

[201] Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms (TALG)*, 6(2):32:1–32:8, 2010.

[202] Shuichi Ueno, Yoji Kajitani, and Shin'ya Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1-3):355–360, 1988.

[203] Baogang Xu, Juan Yan, and Xingxing Yu. Balanced judicious bipartitions of graphs. *Journal of Graph Theory*, 63(3):210–225, 2010.

[204] Baogang Xu and Xingxing Yu. Judicious k-partitions of graphs. *Journal of Combinatorial Theory, Series B (JCTB)*, 99(2):324–337, 2009.

[205] Baogang Xu and Xingxing Yu. On judicious bisections of graphs. *Journal of Combinatorial Theory, Series B (JCTB)*, 106:30–69, 2014.

Thesis Highlight

Name of the Student: Roohani Sharma

Name of the CI/OCC: Dr. Saket Saurabh **Enrolment No.:** MATH10201504012

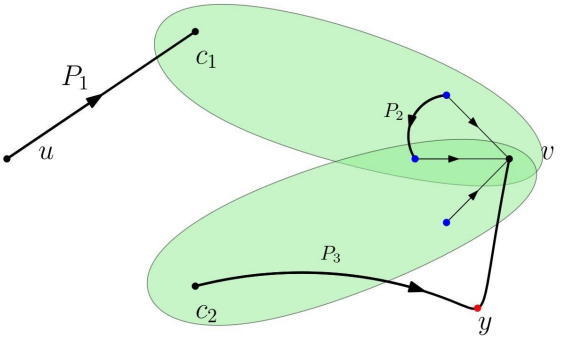
Thesis Title: Advancing the Algorithmic Tool-kit for Parameterized Cut Problems

Discipline: Mathematical Science **Subarea of Discipline:** Theoretical Computer Science

Date of viva voce: 13/08/2020

In this thesis we consider several (di)graph cut problems and study them from the perspective of parameterized complexity and kernelization. The goal of the study is three-fold: first to extend the otherwise limited understanding of parameterized cut problems on directed graphs, second to present novel applications of the rich toolkit available for undirected cut problems and third to develop tools that allow to reuse the algorithms to solve the respective problems in the presence of an additional constraint.

The concrete questions addressed in the thesis are inspired from some major open problems and concerns in the area. Some of these being the famously active open problem of the existence of a polynomial kernel for Directed Feedback Vertex/Arc Set, sub-exponentiality in FPT beyond tournaments, parameterized algorithms for partitioning problems beyond the classical partitioning problems, the existence of single exponential FPT algorithms for stable versions of classical cut problems and the parameterized complexity of Stable Multicut. We address the above questions either in full or extend the results known in literature that take steps to come closer to resolving the actual question. In several cases, our results lead to various enticing insights. In particular, through one of our results, we establish connections between kernelization and fault-tolerant subgraphs. Another result is based on a novel application of important separators in the design of polynomial kernels, which is a rare sight in kernelization. Yet other results give an interesting and useful combinatorial insight about the problem at hand.



The concrete results proved in the thesis are as follows. (1) Directed Feedback Arc Set Directed Edge Odd Cycle Transversal admit a polynomial kernel on digraphs of bounded independence number. (2) Directed Feedback Vertex Set admits a polynomial kernel parameterized by the solution size plus the size of a treewidth-η modulator, for some fixed positive constant η. (3) Out-Forest Deletion and Pumpkin Deletion admits kernels of size $O(k^2)$ and $O(k^3)$ respectively, where k is the solution size. (4) Deletion to DAGs of bounded treewidth admit a polynomial kernel. (5) Out-Forest Deletion, Out-Tree Deletion and Pumpkin Deletion admit algorithms with running time $O^*(2.732^k)$, $O^*(2.562^k)$ and $O^*(2.562^k)$ respectively. (6) Directed Feedback Arc Set, Directed Edge Odd Cycle Transversal, Directed Cutwidth and Optimal Linear Arrangement admit sub-exponential FPT algorithms on digraphs of bounded independence number. (7) Balanced Judicious Bipartition is FPT. (8) Independent Feedback Vertex Set admits an algorithm with running time $O^*(4.1481^k)$, where n is the number of vertices in the input graph and k is the solution size. (9) Stable s-t Separator and Stable Odd Cycle Transversal admit algorithms of running time $2^{k^{\wedge}O(1)}$ (n+m) log n, where n and m is the number of vertices and edges in the input graph, respectively, and k is the solution size. (10) Stable Multicut admit an algorithm with running time $2^{O(k^{\wedge}3)}n^6$ log n.