# Exact Exponential Algorithms & Knot-Free Vertex Deletion

**A Thesis**

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Ajaykrishnan E S


Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

May, 2023

Supervisor: Dr. Soumen Maity and Prof. Saket Saurabh

# Certificate

This is to certify that this dissertation entitled "Exact Exponential Algorithms & Knot-Free Vertex Deletion" towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Ajaykrishnan E S at Indian Institute of Science Education and Research under the supervision of Dr. Soumen Maity, Associate Professor, Department of Mathematics , and Prof. Saket Saurabh, Professor, Department of Computer Science, The Institute of Mathematical Sciences during the academic year 2022-2023.

Dr. Soumen Maity  Prof. Saket Saurabh

Committee:

Dr. Soumen Maity

Prof. Saket Saurabh

Dr. Vivek Mohan Mallick

This thesis is dedicated to my Parents - Mom, Dad, Thank you

# Declaration

I hereby declare that the matter embodied in the report entitled "Exact Exponential Algorithms & Knot-Free Vertex Deletion" are the results of the work carried out by me at the Department of Mathematics, Indian Institute of Science Education and Research, Pune, under the supervision of Dr. Soumen Maity and Prof. Saket Saurabh, and the same has not been submitted elsewhere for any other degree.

Ajaykrishnan E S

# Acknowledgments

I would like to begin by expressing my sincere gratitude to Dr. Soumen Maity for his constant support and guidance. His patient and calm demeanour has made the masters project a delightful experience. I am also grateful to him for introducing me to Prof. Saket Saurabh. The meetings and discussions I had with them played an integral role in the successful conclusion of this project.

I would like to convey my deep gratitude towards Prof. Saket Saurabh for his valuable suggestions and support during the project. His passion for the subject is infectious and never have I walked away from a meeting without being inspired or learning something new.

I wish to take this opportunity to thank Dr. Abhishek Sahu for his time spent on discussions. I would also like to thank all my lab mates and friends - Ajinkya, Hitendra, Mihir and Anirudh - for all the meetings and fun lessons.

I would like to thank all my friends at IISER Pune, things would have been very different had I not found you all. I am especially indebted to Nishu for being a rock-solid anchor in the past five years.

Finally, I would like to express my profound gratitude to my family for their unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you.

# Abstract

A *knot* $K$ in a directed graph $D = (V, E)$ is a strongly connected subgraph of $D$ with at least two vertices, such that there is no arc $(u, v)$ of $D$ with $u \in V(K)$ and $v \notin V(K)$. Given a directed graph $D = (V, E)$, we study Knot-Free Vertex Deletion (KFVD), where the goal is to remove the minimum number of vertices such that the resulting graph does not have any knots. This problem naturally emerges from its application in deadlock resolution in the OR-model of distributed computation. KFVD is known to be NP-complete even on graphs of maximum degree 4. The fastest known exact algorithm in literature for KFVD runs in time $\mathcal{O}^\star(1.576^n)$. We present an improved exact algorithm running in time $\mathcal{O}^\star(1.4549^n)$, where $n$ is the number of vertices in $D$. We also prove that the number of inclusion wise minimal knot-free vertex deletion sets is $\mathcal{O}^\star(1.4549^n)$ and show that this bound is almost optimal by constructing a family of graphs with $\Omega(1.4422^n)$ minimal knot-free vertex deletion sets.

# Contents

# Chapter 1

# Introduction

The paper by Held and Karp [9] in the early sixties, sparked an interest in designing (fast) exact exponential algorithms. In the last couple of decades there has been immense progress in this field, resulting in non-trivial exact exponential algorithms [9, 7, 19, 13, 10, 6, 3]. Alongside optimisation problems, it is also of interest to find exact exponential algorithms for enumeration problems. These are useful in answering natural questions that arise in Graph Theory, about the number of minimal (maximal) vertex subsets that satisfy a given property. In a graph of order $n$ better bounds than the trivial $\mathcal{O}(2^n/\sqrt{n})$ are known only for a few problems. The celebrated result of Moon and Moser [16] proving an upper bound of $\mathcal{O}^\star(1.4422^n)$ for maximal independent sets, alongside that of Fomin et al. for Feedback Vertex Set [9] and Dominating Set [7] are some examples.

## 1.1 Knot-Free Vertex Deletion

A *knot* $K$ in a directed graph $D = (V, E)$ is a strongly connected subgraph of $D$ with at least two vertices, such that there is no arc $(u, v)$ of $D$ with $u \in V(K)$ and $v \notin V(K)$. We call a graph *knot-free* if it does not have any knots. The Knot-Free Vertex Deletion problem belongs to a larger class of problems in which one has to modify the graph via some simple operations such as vertex, edge deletion, addition or contraction to ensure that the resulting graph has some special property. It is formally defined as follows:

| Knot-Free Vertex Deletion |
| --- |
| **Input:** A digraph $D = (V, E)$. |
| **Question:** What is the size of a smallest vertex subset $S$ which satisfies the condition that $G[V \setminus S]$ is knot-free? |

As mentioned previously, this problem naturally emerges from its application in deadlock resolution in the OR-model of distributed computation. A distributed system consists of independent processors which are interconnected via a network which facilitates resource sharing. It is typically represented using a wait-for digraph $D = (V, E)$, where the vertex set $V$ represents processes and edge set $E$ represents wait-conditions. A deadlock occurs when a set of processes wait indefinitely for resources from each other to take actions such as sending a message or releasing a lock. Deadlocks [4] are a common problem in multiprocessing systems, parallel computing and distributed systems.

The AND-model and OR-model are well investigated deadlock models in literature [17]. In the AND-model the process corresponding to a vertex $v$ in the wait-for graph can start only after the processes corresponding to all of its out-neighbours are completed, while in the OR-model $v$ can begin if atleast one of its out-neighbours has finished execution. Deadlocks in the AND-model correspond to cycles and hence deadlock prevention via preempting processes becomes equivalent to the Directed Feedback Vertex Set problem in the wait-for graph. This problem has received considerable attention from various algorithmic viewpoints [5, 19, 14, 15] over the years. Meanwhile, the deadlock resolution in OR-models (which is equivalent to KFVD) has only been explored recently [2, 18].

### 1.1.1 Previous Work

Consider the Directed Feedback Vertex Set problem, where one has to find a smallest size vertex subset whose deletion makes the input digraph acyclic. Observe that acyclic graphs are also knot-free since they do not have strongly connected components of size greater than 1. Hence one way of solving KFVD is to use the solution obtained via an algorithm for Directed Feedback Vertex Set, since the problem has received considerable attention in the past [5, 14, 19, 15]. But the size of a solution for KFVD could be considerably smaller than that of Directed Feedback Vertex Set and hence it is of merit to focus on the

problem directly.

In 2019, Carnerio, Protti and Souza [4] first investigated deadlock resolution in various models via arc or vertex deletion. It was shown that corresponding to the OR-model, the arc deletion problem is polynomial time solvable, while the vertex deletion (KFVD) remains NP-complete even for graphs of maximum degree 4. They further showed that KFVD is solvable in $\mathcal{O}(m\sqrt{n})$ time in graphs of maximum degree 3. The first and only non-trivial exact exponential algorithm for KFVD was presented by Ramanujan et al. [18] and runs in time $\mathcal{O}^\star(1.576^n)$. From the aspect of Parameterised Complexity, Bessy et al. [2] showed that KFVD parameterised by solution size is $W[1]$-hard but FPT parameterised by clique-width. Via [2, 1] and [4], we also know the Fine-Grained complexity of KFVD with respect to both treewidth ($tw$) and size of the largest strongly connected component ($cc$). Under ETH, KFVD does not admit an algorithm running in time $2^{o(tw)}n^{\mathcal{O}(1)}$ but does have an algorithm which runs exponential in tree width. Finally, KFVD has an algorithm running in time $2^{cc}n^{\mathcal{O}(1)}$ but does not have one which can run in $2^{o(cc)}n^{\mathcal{O}(1)}$ assuming SETH

### 1.1.2 Our Contributions

We build upon the work of Ramanujan et al. [18] to obtain the results stated below. Note that the algorithm mentioned in theorem 1.1 is designed using the technique of *Branching* and its complexity is computed via *Measure & Conquer*

**Theorem 1.1.** *There exists an algorithm for* Knot-Free Vertex Deletion *running in* $\mathcal{O}^*(1.4549^n)$. *Furthermore, there is an infinite family of graphs for which the algorithm takes* $\Omega(1.4422^n)$ *time.*

**Theorem 1.2.** *The number of inclusion-wise minimal knot-free vertex deletion sets is* $\mathcal{O}^*(1.4549^n)$ *and there exists an infinite family of graphs with* $\Omega(1.4422^n)$ *many such sets.*

Observe that the upper bound that we prove, is nearly optimal since the existence of the family with $\Omega(1.4422^n)$ many minimal knot-free vertex deletion sets implies that it cannot go lower than $\mathcal{O}^\star(1.4422^n)$. The above results has been concisely summarised and can be found at [21].

## 1.2 Scope of the Thesis

In Chapter 2 we present an overview of the field of Exact Exponential Algorithms and summarise the techniques employed in the following chapters, with examples.

In Chapter 3 we cover the definitions and auxiliary results that are frequently used within the algorithm.

In Chapter 4 we present our algorithm for KFVD, prove the correctness as well as the upper bound for its running time

Finally, in Chapter 5 we prove some complementary results such as a lower bound for the run-time of our algorithm, alongside an upper bound for the number of minimal knot-free vertex deletion sets.

## 1.3 Original Contributions

The thesis presents an exact exponential algorithm for a graph theoretic problem known as Knot-Free Vertex Deletion which is faster than the previous best known exact algorithm in literature. The algorithm alongside the proof for its correctness and upper bound for run-time can be found in Chapter 5. Some results complementing the algorithm, such as a lower bound for its run-time, alongside some original combinatorial results are presented in Chapter 6. Both the above mentioned chapters are entirely original. Alongside this, we prove some original lemmas that are necessary to prove the correctness of our algorithm, in Chapter 4. Consult [21] for a more concise exposition of the results.

# Chapter 2

# Preliminaries

This chapter contains a brief summary of the common definitions and concepts that we require from the field of Graph Theory and Computational Complexity. It is a concise literature review meant to prepare the reader for the rest of the thesis.

## 2.1 Graph Theory

A graph is a widely studied combinatorial structure with $V$ the set of vertices and $E \subseteq V \times V$ the set of edges. It is used to model real world scenarios with vertices representing objects and edges representing some relation between said objects. A directed graphs is a type of graph where the relations are represented by ordered pairs called in/out edges or arcs. In this thesis, we deal mainly with directed graphs since the central problem that we work on is defined on directed graphs.

Given an arc $e = (u, v)$, we refer to $e$ as an out-edge of $u$, and an in-edge of $v$. Similarly, $u$ is an in-neighbour of $v$ and consequently $v$ is an out-neighbour of $u$. We denote the in-neighbours of a vertex $x$ by $N^-(x)$ and out-neighbours by $N^+(x)$. A vertex without any in-neighbour is called a source vertex while one without out-neighbours is called a sink vertex. A path $P_{uv}$ from $u$ to $v$ in $D$ is a sequence of vertices $(v_1, v_2, \ldots v_p)$ where $v_1 = u, v_p = v$ and $(v_i, v_{i+1})$ is an arc in $D$ for every $i < p$.

A strongly connected component $Q$ is a subset of vertices such the given $v, u \in Q$, there is a directed path from $u$ to $v$ and vice versa. A graph $H$ is called a subgraph of a graph $G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A graph $H$ is called an induced subgraph of graph $G$, if $V(H) \subseteq V(G)$ and $E(H)$ is set of all the edges of $G$ with both the endpoints in $V(H)$. For a set of vertices $A \subseteq V(G)$ the induced subgraph on $A$ is represented by $G[A]$.

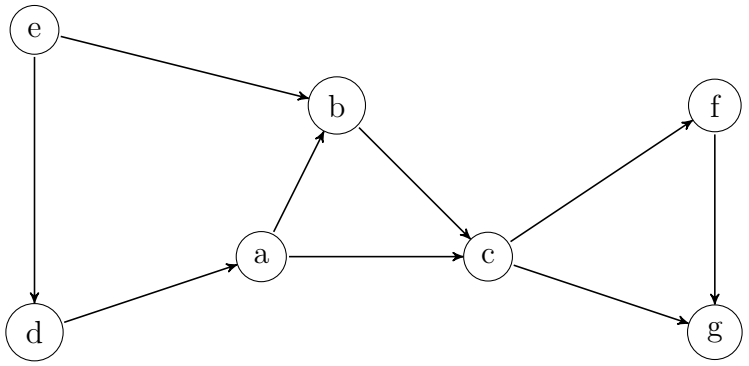


Figure 2.1: Digraph $D$

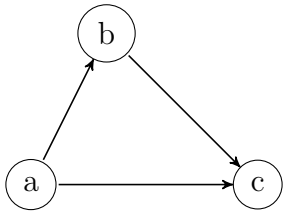


Figure 2.2: Subgraph $H$

Observe that in Figure 2.1, $(a, b)$ is an arc. It is an out-edge for $a$ and an in-edge for $b$. Consequently $a$ is an in-neighbour of $b$ and $b$ is an out-neighbour of $a$. $e$ is a source vertex while $g$ is a sink vertex and $(e, b, c, g)$ denotes a directed path of length 4 from $e$ to $g$. Finally Figure 2.2 is a subgraph of the graph in Figure 2.1 induced by the vertex set $\{a, b, c\}$.

## 2.2 Computational Complexity

A language is a set of finite strings $\Sigma^\star$ over some alphabet $\Sigma$. A computational problem $P$ is a language over some $\Sigma$ and x is said to be a yes-instance of $P$ if and only if $x \in P$.

There are various types of computational problems such as decision problems, optimization problems and enumeration problems. Complexity theory concerns itself with the study of resources required to solve such problems. The main quantity used to classify the hardness of a problem is the number of basic operations such as addition, multiplication, logical OR and logical AND required to solve it. A problem is said to be solvable in time $T(n)$ if it can be solved using at most $T(n)$ basic operations, where $n$ is the size of the input. In order to concisely denote the time complexities, we use the following notation. Given two functions $f, g$ defined on $\mathbb{N}$ we denote $f \in \mathcal{O}(g)$ if there exists some positive real constant $c$ such that $f(n) \leq c * g(n)$ for every $n$. Similarly, $f \in \mathcal{O}^{\star}(g)$ if there exists some positive real constant $c$ and some polynomial *poly* such that $f(n) \leq c * g(n) * poly(n)$ for every $n$. Now, we define classes of computational problems below, based on the time it takes to solve and verify their solutions.

**P** is the class of problems which can be solved in polynomial time.

**NP** is the class of problems for which there exists an algorithm which can verify solutions in polynomial time.

These classes satisfy the relation P $\subseteq$ NP. The question whether P = NP is one of the oldest and largest question in computational complexity. The quest to answer this question has led to a lot of deep and insightful research into the nature of computational problems.

### 2.2.1 NP-completeness

NP-complete problems are the hardest problems in NP. They do not currently have polynomial time or *efficient* algorithms and at the same time, we do not have machinery to prove that there will never be efficient algorithms for these problems. Consider a problem $\Pi_1$, we prove that it is NP-complete by showing that if there exists an algorithm that can solve $\Pi_1$ efficiently, then we can construct another algorithm which can solve $\Pi_2$, where the latter is a problem which is proven to be NP-complete.

**Definition 2.1.** *[12]* ***Polynomial-Time Reductions:*** *Let $A, B$ be two decision problems. We say that $A$ reduces to $B$, denoted by $A \leq_P B$ if there exists a polynomial time computable function which takes an input instance $x$ of $A$ and converts it into an instance $f(x)$ of $B$ such that $x$ is a Yes instance of $A$ if and only if $f(x)$ is a Yes instance of $B$.*

Observe that if $A \leq_P B$ then $B$ is harder than $A$ in the sense that the existence of a polynomial time algorithm for $B$ implies the existence of a polynomial time algorithm for $A$ via this reduction.

**Definition 2.2.** *[12]* ***NP-hard:*** *A decision problem $A$ is NP-hard if $B \leq_P A$ for every $B \in NP$.*

**Definition 2.3.** *[12]* ***NP-complete:*** *A decision problem $A$ is NP-complete if $A \in NP$ and $A$ is NP-hard.*

**Theorem 2.1.** *[12] If $X$ is an NP-complete problem, then the existence of a polynomial time algorithm for $X$ implies $P = NP$*

**Theorem 2.2.** *[12] If $Y$ is NP-complete and $X$ is a problem in NP satisfying $Y \leq_P X$ then $X$ is also NP-complete*

Hence due to Theorem 2.1 unless P=NP, there cannot be an algorithm for any NP-complete problem, which solves *every* instance *correctly* and *efficiently.*

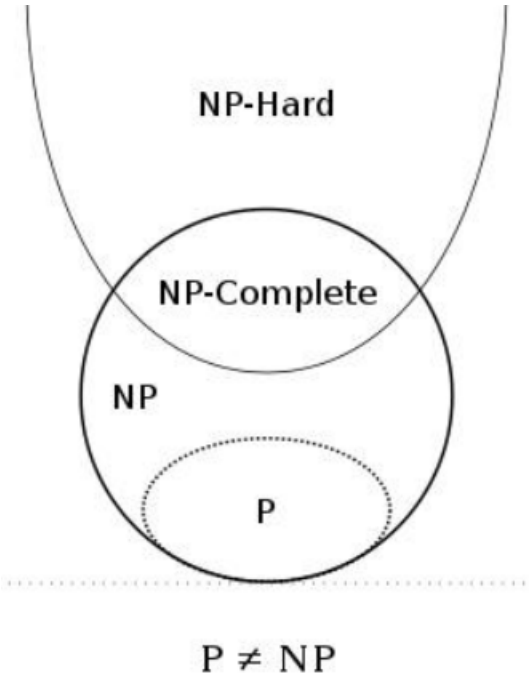


Figure 2.3: The hierarchy of computational problems

There are a multitude of ways to deal with this intractability. We can compromise on any of the above mentioned properties while trying to maintain the others. Compromising correctness leads to approximation algorithms, weakening the requirement to solve every instance leads to parameterised algorithms and algorithms for special classes of input. Finally, we have exact exponential algorithms which we get by letting go of efficiency or the need for a polynomial time algorithm.

# Chapter 3

# Exact Exponential Algorithms

This chapter presents *branching* and *measure & conquer* which are two common and powerful techniques in Exact Exponential Algorithms. It is a literature review meant to equip the reader with an understanding of the technique used in the design of our algorithm for KFVD.

An exact exponential algorithm for an optimization problem $\Pi$ takes an input of size $n$ and returns an optimal solution in time which is exponential in $n$. The construction of such algorithms are of interest for a variety of reasons, some of which are

- Exponential time algorithms are sometimes the best one can hope for. This is due to the widely believed hypothesis of P $\neq$ NP under which the large class of NP-complete problems cannot be solved in polynomial time.
- These algorithms provide insight into the problem structure which has a variety of applications in graph theory and other algorithmic regimes. An example would be the use of enumeration algorithms to prove upper bounds on the number of such structures in graphs.
- Technological advances can help speed up the execution of algorithms but this increases the size of solvable instances by an additive factor. Meanwhile any improvement made via algorithm design increases the size of solvable instances by a multiplicative factor.

The early works on designing such algorithms was surveyed and summarised by Gerhard Woeginger [22] which inspired an elaborate monograph by Fomin and Kratsch titled Exact

Exponential Algorithms [8]. Amongst the wide range of techniques for designing such algorithms, two popular ones are branching and measure & conquer, which we explore in the following sections.

## 3.1 Branching

Branching is a fundamental algorithmic technique to design fast exponential algorithms by decomposing a given instance into smaller ones and using their solution to build a solution for the original instance.

Branching algorithms apply a combination of *reduction rules* and *branching rules*. Reduction rules simplify the problem instance and branching rules split the given instance into smaller instances. Branching algorithms are convenient to use since they typically only need polynomial space and can be naturally improved by looking for finer and more involved branching rules. A well constructed branching algorithm must have branching rules which account for all possible cases that need to be considered and reduction rules which are provably true. The typical argument is that there is at least one optimal solution which is not overlooked by the algorithm. More often than not, the rules are straightforward enough to not be explicitly proven.

An important aspect of any branching algorithm is the analysis of its worst case time complexity. Usually, such algorithms have reduction rules which can be executed in polynomial time. Further, the branching rules are such that the solution to the smaller instances can be combined efficiently to get a solution to the original instance. Under these assumptions, the time it takes for the algorithm to execute is proportional to the total number of smaller instances that need to be solved, counting multiple occurrences. It is beneficial to consider the *search tree* of an execution of the algorithm to better understand this concept. It is constructed by assigning a node to the input instance, and for every smaller instance obtained via a branching rule we assign a child node to the node (corresponding to the instance) on which the rule was applied. Note that we do not assign nodes for instances obtained via reduction rules. It follows from the construction that the number of subproblems solved for any problem instance is equal to the number of nodes in the search tree, which can be bounded by counting the number of leaves. Let us denote the maximum number of leaves for an input instance of size $n$ by $T(n)$. Consider a branching rule which takes this instance

and decomposes it to $r$ smaller instances, $j^{th}$ of which is of size $n - t_j$. We call $(t_1, t_2, \ldots, t_r)$ the branching vector of the rule. The reduction in size implies a linear recurrence of the form

$$T(n) = T(n - t_1) + T(n - t_2) + \cdots + T(n - t_r) \tag{3.1}$$

This can be solved via many well-known techniques [20]. Observe that the worst case running time of the branching algorithm is the maximum amongst the worst case run-time of its branching rules.

A fact when it comes to linear recurrences such as equation 3.1 is that its base solution is always of the form $c^n$ for some complex number $c$. Furthermore, $c$ has to be a root of the polynomial equation

$$x^n - x^{n-t_1} - x^{n-t_2} - \ \ldots \ - x^{n-t_r} = 0 \tag{3.2}$$

Since we are only interested in the worst case running time of the algorithm, we obtain $T(n) = c^n$ where $c$ is the largest positive root of equation 3.2.

### 3.1.1 Example: Vertex Cover

VERTEX COVER

**Input:** A graph $G = (V, E)$.

**Question:** What is the size of a smallest vertex subset $S$ that satisfies $u \in S$ or $v \in S$ for every $(u, v) \in E$?

VERTEX COVER is arguably the most widely studied graph problem in history. It was proven to be NP-complete at a very early stage by Richard Karp, and it belongs to his famous list of 21 NP-complete problems [11]. It has been attacked via a variety of methods ranging from approximation to parameterization and it responds positively to most methods. Note that, since it is a vertex subset problem, the brute force method of checking every subset takes $\mathcal{O}^\star(2^n)$ time. Consider the following graph in which the shaded vertices make up a solution to the vertex cover problem. Observe that every edge is incident upon at least one shaded vertex. Furthermore, since $\{(e, d), (a, b), (c, g), (h, f)\}$ is a set of disjoint edges (maximum matching), we need atleast 4 vertices to form a vertex cover. Hence the shaded solution is an optimal solution.

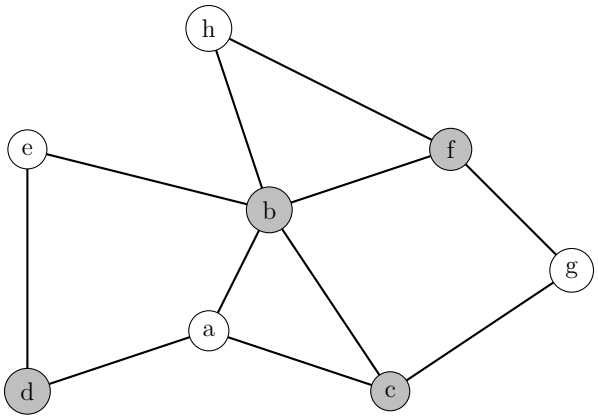


Figure 3.1: Example instance of vertex cover

Now we make a few observations which will be useful in designing a branching algorithm.

- Isolated vertices do not cover any edge and hence is not present in any optimal solution.
- Vertices of degree 1 only covers one edge which is also covered by its neighbour. If there is an optimal solution which contains such a vertex, then removing it and adding its neighbour still leaves us with a vertex cover of (at most) same size. Hence if the input instance has a degree 1 vertex, there is always a solution which contains its neighbour.
- If a vertex is not present in an optimal solution, then all of its neighbours must be in order to cover all the edges adjacent to it.

**Algorithm 1:** $\mathsf{VC}(G)$

**Input:** A graph $G$
**Output:** The size of a smallest vertex cover

```
1 if V(G) = ∅ then
2 |   return 0
3 if ∃ x such that N(x) = ∅ then
4 |   return VC(G − {x})
5 if ∃ x such that N(x) = {y} then
6 |   return VC(G − {y}) + 1
7 else
8 |   pick x ∈ V(G) with maximum degree
9 |   return min{ VC(G − x) + 1, VC(G − N[x]) + |N(x)| }
```

Note that lines 1-2 of Algorithm 1 takes care of the base case where the input graph is empty and 3-4 is a reduction whose correctness follows from our first observation. Similarly lines 5-6 follows from the second one, while 7-9 denote a branching rule arising from the final observation. Observe that given any input at least one of the rules can be applied and each of the rules creates instances of smaller size (or terminates the algorithm). This alongside the correctness of each individual rule proves that the algorithm finds an optimal solution in finite time.

Now, in order to compute the worst case running time of Algorithm 1, we need to find branching vectors for each of the branching rules. For Algorithm 1 we note that there is only one branching rule, corresponding to lines 6-8. In it, we branch on a vertex ($x$) of degree at least 2, since vertices of degree 0 and 1 are removed. One branch corresponds to the possibility that $x$ is in the optimal vertex cover, while the other considers the possibility that it isn't. In the first one, the vertex $x$ is removed giving a drop of at least 1 while in the other, the neighbourhood of $x$ can be removed giving a drop of at least 3. Hence the only branching vector corresponding to Algorithm 1 is $(1, 3)$. This implies the recurrence relation

$$x^3 - x^2 - 1 = 0.$$

This can be solved to find the largest positive root $c$ which turns out to be less than 1.4657. Hence Algorithm VC runs in time $\mathcal{O}^\star(1.4657^n)$.

## 3.2 Measure & Conquer

Measure and conquer is a powerful technique used to better analyse the worst case running time of branching algorithms. It is motivated by the fact that the naive analysis of branching uses only "natural parameters" such as number of vertices to account for the drop in size of the instance for each branch. Conceptually, the idea of measure and conquer is not very different from the earlier simple analysis. The only difference being that instead of using such simple parameters to quantify the drop, we can choose the *measure* more freely and cleverly. The following are some points to keep in mind while designing a measure for any branching algorithm.

- Every instance of the problem with non-positive measure must be solvable in polynomial time. This allows us to use the same concepts that were used earlier, since $T(\mu) = n^{\mathcal{O}(1)}$ whenever $\mu \leq 0$.
- Measure of every smaller instance created via branching or reduction must have a smaller measure than the original instance. This is to ensure that if we begin with a positive measure, the measure reaches zero after a finite number of branching and reduction rules are applied.
- We must be able to upper bound the measure of the input instance by some linear function of the "natural parameter". In the case of graphs, we require that $\mu(G) \leq f(n)$ for some function of $n$. This allows us to translate the time complexity obtained in terms of some non-standard measure $\mu$ to one in terms of the natural parameter of the input instance, which lends itself to better comparisons and statements.

Note that the procedure after defining a measure ($\mu$) which satisfies these requirements are identical to that of the earlier analysis. For every branching rule, we quantify the drop in measure and find the branching vector $(t_1, t_2, \ldots, t_r)$. This implies the recurrence relation

$$T(\mu) = T(\mu - t_1) + T(\mu - t_2) - \ldots - T(\mu - t_r) \tag{3.3}$$

This is solved to obtain a solution of the form $T(\mu(G)) \leq c^{\mu(G)} \leq c^{f(n)}$ giving us an upper bound for the time taken on the input instance $G$.

### 3.2.1 Example: Independent Set

| Independent Set |
| --- |
| **Input:** A graph $G = (V, E)$. |
| **Question:** What is the size $\alpha(G)$ of a largest vertex subset $S$ such that the graph induced by $S$ has no edges? |

Independent Set is closely connected to vertex cover (see Section 3.1.1), due to the fact that if $S$ is a largest independent set in $G$, then $V(G) \setminus S$ is a smallest vertex cover. Note that, since this is also a vertex subset problem, the brute force method of checking every

subset takes $\mathcal{O}^\star(2^n)$ time. Consider the following graph in which the shaded vertices make up a solution to the independent set problem. Observe that the shaded solution is optimal, since the existence of a larger solution for independent set implies the existence of a smaller solution to the vertex cover problem for the same graph which doesn't exist via the proof sketch in Section 3.1.1.

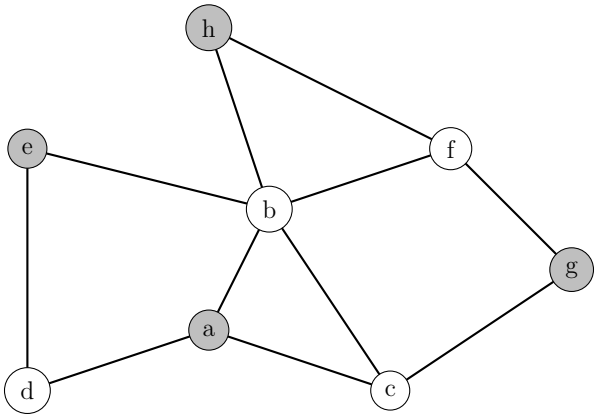


Figure 3.2: Example instance of independent set

Now let us make a few observations which will be useful in designing a branching algorithm.

- Isolated vertices are present in every optimal solution. This is due to the fact that by definition they do not have any neighbours and can hence be freely added to any independent set.
- Given a vertex $v$ of degree 1, we can always find an optimal solution which contains it. This follows from the fact that any optimal solution which does not contain $v$, contains its neighbour (if it does not, we can freely add $v$ and get a larger independent set). Hence we can pick any optimal solution and if it does not contain $v$, we can replace its neighbour with $v$ and still maintain an optimal solution.
- If a vertex is present in an optimal solution, then none of its neighbours can be, since otherwise the subgraph induced by them will contain an edge.
- If the degree of every vertex in the graph is 2, then it is a disjoint union of cycles. In any cycle graph, the size of $\alpha(G) = \lfloor \frac{n}{2} \rfloor$. This can be used to solve the problem in polynomial time, if the maximum degree of the graph is 2.

The idea for the algorithm is to build reduction rules using the initial observations, branch on high degree vertices using ideas from observation 3 and finally when the maximum degree of the graph is small enough, apply reduction rules using observation 4. We shall analyse the algorithm initially via the naive method and later via two different measures.

**Algorithm 2:** $\mathsf{MIS}(G)$

**Input:** A graph $G$
**Output:** The size of a largest Independent Set

```
1  if V(G) = ∅ then
2  |  return 0
3  if ∃ x such that N(x) = ∅ then
4  |  return MIS(G − {x}) + 1
5  if ∃ x such that N(x) = {y} then
6  |  return MIS(G − {N[x]}) + 1
7  else
8  |  if Δ(G) ≥ 3 then
9  |  |  pick x ∈ V(G) with maximum degree
10 |  |  return max{ MIS(G − x), VC(G − N[x]) + 1 }
11 |  else
12 |  |  compute α(G) in polynomial time
13 |  |  return α(G)
```

Observe that the algorithm has only one branching rule, corresponding to lines 8-10. Let us denote the branch where the vertex $x$ on which we branch, is added to the solution set by $IN$ and the other branch by $OUT$. We note that in $IN$ the whole neighbourhood of $x$ can be removed while in $OUT$ only $x$ can be. Since the degree of $x$ is at least 3 we get a drop in size of at least 4 in $IN$ and 1 for $OUT$. This gives a branching vector $(1, 4)$ which can be evaluated by solving the corresponding polynomial equation to obtain a worst case running time that is $\mathcal{O}^\star(1.3803^n)$. Hence the naive analysis gives the worst case time complexity for $\mathsf{MIS}$ as $\mathcal{O}^\star(1.3803^n)$ which we would like to improve.

We begin with the observation that vertices of low degree $0 - 2$ seem to be easier to handle, since we have reduction rules which can get rid of them. Following this intuition, let us define a measure for an input graph $G$ with $\mu(G) = v_{\geq 3}$, where $v_{\geq 3}$ denotes the number of vertices of degree 3 or higher. Note that this is equivalent to setting the weight of vertices

with degree 2 or lower as 0 and that of every other vertex as 1. Now in $OUT$, we get a drop in measure of 1 since a vertex of degree higher than 3 is removed. But in $IN$, even though all the neighbours of $x$ is removed, we cannot count them towards the drop in measure, since they could be degree 2 vertices. Hence the branching vector is $(1, 1)$ which solves to give $T(\mu) \leq 2^{\mu(G)} = \mathcal{O}^\star(2^n)$ which is worse than the "naive" analysis.

Finally, let us try a slightly different measure $\mu(G) = v_{\geq 3} + \frac{v_2}{2}$ where $V_{\geq 3}$ denotes the number of vertices of degree at least 3 and $v_2$ denotes the number of vertices of degree 2. Here, we set weight of vertices with degree more than 2 as 1, vertices of degree 2 has weight 0.5 and vertices of lower degree have weight 0. We observe that if $x$ has neighbours of degree 2, then in $OUT$ the degree of that vertex becomes 1 which contributes to the drop in measure. Hence the contribution of each neighbour to the potential drop over both the branches is at least 1. This implies, $OUT \geq 1$, $IN \geq 1$ and finally $OUT + IN \geq 2 + d(v)$, since $x$ contributes 1 in both the branches. Here there are two possibilities, one is that the vertex $x$ has degree 4 or higher. The worst case branching vector possible under this possibility is $(1, 5)$ which evaluates to $T(\mu) \leq 1.3248^{\mu(G)} = \mathcal{O}^\star(1.3248^n)$. The other is that the maximum degree of the graph is 3 under which possibility, every neighbour of $x$ contributes 0.5 to both the branches. Hence $OUT, IN \geq 1 + \frac{d(v)}{2} = 2.5$ which evaluates to $\mathcal{O}^\star(1.3196^n)$

Observe that by simply changing the weight assigned to some vertices we were able to reduce the upper bound in running time from $\mathcal{O}^\star(1.3803^n)$ to $\mathcal{O}^\star(1.3248^n)$. Note that here there is no reason why setting weight of degree 2 vertices as 0.5 is the optimal. In fact, setting weight of degree 2 as 0.5966 and degree 3 as 0.9287 while keeping the rest same, gives an improved bound of $\mathcal{O}^\star(1.2905^n)$.

# Chapter 4

# The Knot Free Vertex Deletion problem

In this chapter, we present some known ideas and reduction rules which are used in our algorithm. Alongside this summary, we also state and prove some original lemmas [21] which draw inspiration from [18] and make some new definitions to help concisely present our algorithm.

Recall that a *strongly connected subgraph* in a directed graph $D$ is a subgraph $K$ such that for every $u, v \in V(K)$, there exists a directed path in $K$ from $u$ to $v$ and from $v$ to $u$. A *knot* $K$ in a directed graph $D = (V, E)$ is a strongly connected subgraph of $D$ with at least two vertices, such that there is no arc $(u, v)$ of $D$ with $u \in V(K)$ and $v \notin V(K)$. The knot-free vertex deletion problem is defined in Section 1.1 as follows:

| Knot-Free Vertex Deletion |
| --- |
| **Input:** A digraph $D = (V, E)$. |
| **Question:** What is the size of a smallest vertex subset $S$ which satisfies the condition that $G[V \setminus S]$ is knot-free? |

Figure 4.1 provides an example of an instance of the knot-free vertex deletion problem, with the highlighted vertices forming a solution. Figure 4.2 is the knot-free digraph obtained after deleting the solution vertices.

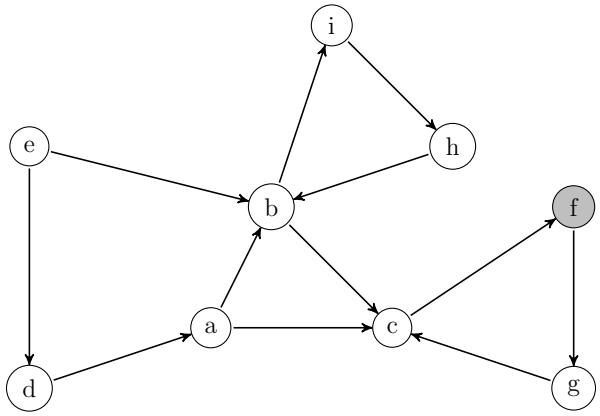


Figure 4.1: Example instance of knot-free vertex deletion problem

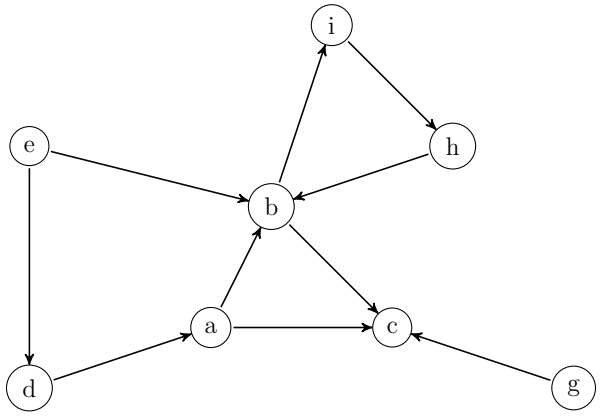


Figure 4.2: Knot-free graph after removal of solution set

Observe that in the input graph, there is one knot $\{c, f, g\}$ and atleast one vertex needs to be deleted to ensure that this knot is removed. Further, after the deletion of $\{f\}$ observe that the earlier knot is removed and there aren't any strongly connected components of size 2 or higher without out-edges.

In a 2019 paper by Stéphane Bessy et al. [2], a very useful characterisation of knot-free graphs were proven. This definition and its corrollary will play a crucial role in the design of our algorithm.

**Proposition 4.1.** [2] *A digraph $D$ is knot-free if and only if for every vertex $v$ of $D$, $v$ has a path to a sink.*

*Proof.* Consider any vertex $v$ in a knot-free digraph $D$. We shall construct a path from $v$ to some sink vertex in $D$. If $v$ is not a sink, then it belongs to a strongly connected component $Q$ with an out edge. Consider a path beginning with $v$ and going out of $Q$, reaching a vertex $v'$. Observe that we cannot go from $v'$ to any other vertex of $Q$. We can repeat this process considering $v'$ as the starting vertex. But, every time this process is repeated, the vertices that are reachable from the current vertex keeps dropping. Since there are only finitely many vertices, we arrive at a vertex $u$ from which we cannot reach any more vertices. $u$ has to be a sink vertex, as any out-neighbours $u$ has, can be reached from $u$.
Consider a digraph $D$ such that every vertex of $D$ has a directed path to some sink in $D$. Pick any strongly connected component $Q$ with two or more vertices. Observe that the vertices of $Q$ must also have a path to some sink, and that the sink cannot be a part of an strongly connected component of size more than one. Hence, there must be some edge going out of $Q$ and consequently the digraph $D$ is knot-free. □

**Corollary 4.2.** [2] *For any minimal solution $S \subseteq V(D)$ with the set of sink vertices $Z$ in $D - S$, we have $N^+(Z) = S$.*

*Proof.* Observe that, given the set of sinks $Z$ corresponding to any minimal solution $S \subseteq V(D)$, the set $N^+(Z)$ forms a solution. Removing the out-neighbours of $Z$ makes them sinks, and this set is enough to ensure that every vertex has a path to some sink. Hence via Proposition 4.1 we get the relation $N^+(Z) = S$. □

The above propositions ensure that to solve the Knot-Free Vertex Deletion problem we only need to find the set of vertices $Z$ such that upon deleting its out-neighbours the graph is knot-free and $|N(Z)|$ is minimum. Therefore, rather than directly identifying the solution vertices our algorithm finds the set of sink vertices in the optimal solution, hence indirectly finding the solution vertices.

Observe that in the example provided in Figure 4.1 the highlighted vertex is the out-neighbours of the vertex $c$ which becomes a sink in Figure 4.2. Hence by removing $N^+(c) = \{f\}$ we get a knot-free graph in which every vertex can reach $c$ via some directed path.

## 4.1 Definitions

**Strategy of our Algorithm.** The algorithm expands on the ideas used in [18] where the algorithm branches on the possibility that a vertex $v \in V(D)$ is either a sink or a non-sink vertex in some optimal solution. In the branch where we conclude $v$ to be a non-sink vertex there are two possibilities, $v$ is either in the deletion set or not. To track this additional information that $v$ is non-sink, we use a potential function $\phi$ for $V(D)$ defined as follows.

**Definition 4.1** (Potential function)**.** *Given a digraph $D = (V, E)$, we define a* potential function *on $V(D)$, $\phi : V(D) \to \{0.25, 1\}$ such that $\phi(v) = 1$, if $v$ is a potential vertex to become a sink in an optimal solution and $\phi(v) = 0.25$, if $v$ is a non-sink vertex. For any subset $V' \subseteq V(D)$, $\phi(V') = \sum_{x \in V'} \phi(x)$. We call a vertex $v$ as an* undecided *vertex if $\phi(v) = 1$ and a* semi-decided *vertex if $\phi(v) = 0.25$.*

**Definition 4.2** (Feasible solution)**.** *A set $S \subseteq V(D)$ is called a feasible solution for $(D, \phi)$ if $D - S$ is knot-free and for any sink vertex $s$ in $D - S$, $\phi(s) = 1$. $\mathsf{KFVD}(D, \phi)$ denotes the size of an optimal solution for $(D, \phi)$.*

Given an input digraph $D$, we set the potential of all the vertices as 1 and run the algorithm. Over the course of the algorithm, we change the potential of vertices which are not sinks in the optimal solution to 0.25. Now we make some definitions which will be useful for presenting our algorithm.

**Definition 4.3** (In-reachability set)**.** *The* in-reachability set *of a vertex $v$ denoted by $R^-(v)$, is the set of vertices that can reach $v$ via some directed path in $D - N^+(v)$.*

**Definition 4.4** (Out-reachability set)**.** *The* out-reachability set *of a vertex $v$ denoted by $R^+(v)$, is the set of* undecided *vertices (say $u$) that can be reached from $v$ via some directed path in $D - N^+(u)$.*

Notice that $v \in R^-(v)$ and $u \in R^+(v)$ if and only if $v \in R^-(u)$. We denote the set $N^+(v) \cup R^-(v)$ as $R(v)$. We also use $S_{min}$ to denote a minimal solution for the given instance $(D, \phi)$ and $Z_{min}$ for the set of sinks in $D - S_{min}$. Similarly, we use $S_{opt}$ and $Z_{opt}$ to refer to an optimal solution and its sink set. We further define an update function Algorithm

3 below.

**Algorithm 3:** *update*$(D, \phi, s, NS)$

**Input:** A directed graph $D$, a potential function $\phi$, a sink vertex $s$ (optional) and a set of non-sink vertices $NS$ (optional)

**Output:** An updated digraph $D'$ and an updated potential function $\phi'$

```
1 D' = D, φ' = φ
2 if input s is provided then
3 |   D' = D' − R(s)
4 |   for v ∈ R+(s) do
5 |   |   φ'(v) = 0.25
6 if input NS is provided then
7 |   for v ∈ NS do
8 |   |   φ'(v) = 0.25
9 return D', φ'
```

In the example given below, the vertices which are coloured *yellow* are *semi-decided*. Using this, we observe that $R^-(h) = \{i\}$ while $R^+(h) = \{c\}$. Note than even though the vertices from the set $\{e, d, b, a\}$ can reach $h$ in $D$, they do not belong to $R^-(h)$ since after deleting the out-neighbour of $h$ they can no longer reach $h$. Similarly even though $h$ can reach any vertex from the set $\{b, f\}$, they do not belong to $R^+(h)$ since they are semi-decided.

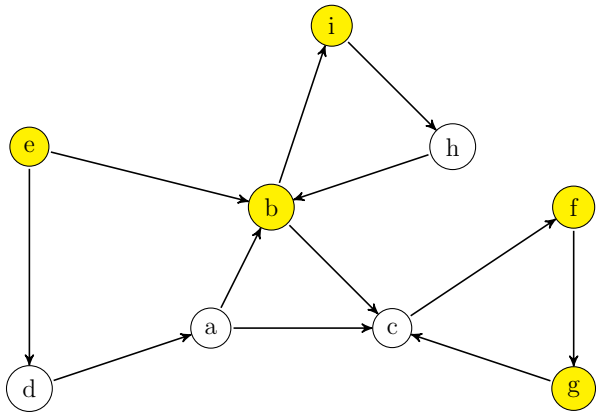


Figure 4.3: Example instance of knot-free vertex deletion problem

## 4.2 Auxiliary Results

The following rules were used by Ramanujan et al. in 2022 to design the first exact exponential algorithm for KFVD, and shall also be implemented in our algorithm.

**Reducion Rule 1.** [18] *If all the vertices in $D$ are semi-decided and $D$ has no source or sink vertices, then $V(D)$ is contained inside any feasible solution for $(D, \phi)$.*

**Reducion Rule 2.** [18] *Let $v \in D$ be such that $N^-(v) = \emptyset$, $S$ is an optimal solution for $D$ iff $S$ is an optimal solution for $D' = D - v$.*

**Reducion Rule 3.** [18] *Let $v \in D$ be such that $N^+(v) = \emptyset$, $S$ is an optimal solution for $D$ iff $S$ is an optimal solution for $D' = D - R(v)$.*

With this, we are ready to prove some propositions which are required to prove the correctness of various subroutines in our algorithm.

**Proposition 4.3.** *If $x \in Z_{min}$, then we have, $N^+(x) \subseteq S_{min}$, $S_{min} \cap R^-(x) = \emptyset$ and $Z_{min} \cap R^+(x) = \emptyset$.*

*Proof.* By the definition of a sink vertex, if $x \in Z_{min}$ then $N^+(x)$ is in $S_{min}$. Let $Y = S_{min} \cap R^-(x)$. We claim $S' = S_{min} \setminus Y$ is also a solution which will contradict the fact that $S_{min}$ is a minimal solution. Suppose $S'$ is not a solution, then there exists a vertex $v$ in $D - S'$, that does not reach a sink in $D - S'$ but $v$ reaches some sink $s$ in $D - S_{min}$. Since every vertex in $Y$ reaches sink $x$ in $D - S'$, $v \notin Y$. If $s$ is not a sink vertex in $D - S'$, then some vertex $y \in Y$ is an out-neighbor of $s$. But then $v$ can reach the sink $x$ in $D - S'$ via $y$. Hence, $S'$ is a solution of size strictly smaller than $S_{min}$, which is a contradiction. Since $S_{min} \cap R^-(x) = \emptyset$, we also have $Z_{min} \cap R^-(x) = \emptyset$. Now, let $s \in R^+(x)$. If $s \in Z_{min}$ then by definition of $R^+(x)$, $x \in Z_{min} \cap R^-(s)$ which is a contradiction. □

**Proposition 4.4.** *If $x \in Z_{min}$, then $S_{min} = N^+(x) \cup S'$, where $S'$ is a minimal feasible solution for $(D', \phi') = update(D, \phi, x, -)$. Also, if $S'_{min}$ is a minimal solution for $(D', \phi')$ then $S = N^+(x) \cup S'_{min}$ is a solution for $(D, \phi)$.*

*Proof.* First, assume that $S'_{min}$ is a minimal solution for $(D', \phi')$. We claim that $S = S'_{min} \cup N^+(x)$ is a solution for $(D, \phi)$. Suppose not, then there exists a vertex $v$ that does not reach a sink in $D - S$. Note that $v \notin R^-(x)$ as all vertices in $R^-(x)$ can reach sink $x$ in $D - S$. Then $v \notin R(x) \cup S$ and hence it is also in $D' - S_{min}$, where it reaches a sink $s$. Since this path is disjoint from $S'_{min} \cup R^-(x)$, $v$ still can reach $s$ via the same path in $D - S$. Note that $s$ is not a sink in $D - S$ only if it has an out-neighbor in $R^-(x) \setminus N^+(x)$. But then $s$ and $v$ are in $R^-(x)$ which is a contradiction. Hence, $v$ can still reach the same sink $s$ and $S = S'_{min} \cup N^+(x)$ is a solution for $D$.

Now, we prove that $S' = S_{min} \setminus N^+(x)$ is a solution for $(D', \phi')$. Since, $S_{min}$ is optimal, $S' \cap R^+(x) = \emptyset$ via Claim 4.3 and thus, $S'$ is feasible. If $S'$ is not a solution, then there has to be a vertex $v$ in $D' - S'$ that does not reach any sink. Since $S_{min} \subseteq R(x) \cup S'$, $v \notin S_{min}$. But $S_{min}$ is a solution for $(D, \phi)$ and $v$ can reach some sink $s \in Z_{min}$ in $D - S_{min}$. Note that $s \neq x$, since $v$ is not in $R^-(x)$ and cannot reach $x$ in $D - S_{opt}$. Then $v$ has a path to $s$ which is disjoint from $S_{opt} \supseteq N^+(x)$. Moreover this path is also disjoint from the set $R(x) \setminus N^+(x)$, since $v \notin R(x)$. Hence this path is disjoint from $R(x)$ and $S_{opt}$. Therefore, $v$ can reach $s$ via the same path in $D' - S'$. If $s$ is a sink in $D - S_{opt}$, then $s$ is also a sink in $D' - S'$. Hence, $v$ has a path to a sink in $D' - S'$, which is a contradiction. It implies that $S' = S_{min} \setminus N^+(x)$ is a solution for $(D', \phi')$.

Finally, assume $S' = S_{min} \setminus N^+(x)$ is not a minimal solution for $(D', \phi')$. Then there exists $S'' \subset S'$ which is also a solution for $(D', \phi')$, but then $S'' \cup N^+(x)$ is also a solution for $(D, \phi)$ contradicting the minimality of $S_{min}$. □

**Corollary 4.5.** *If $x \in Z_{opt}$, then we have, $|S_{opt}| = |N^+(x)| + \mathsf{KFVD}(D', \phi')$, where $(D', \phi') = update(D, \phi, x, -)$.*

*Proof.* Let us assume $S'_{opt}$ to be an optimal solution to $(D', \phi')$. By Proposition 4.4, $S = N^+(x) \cup S'_{opt}$ is a solution to $(D, \phi)$. Now, if $(D, \phi)$ has a minimal solution $\tilde{S}$ smaller than $S$ then via Proposition 4.4, $\tilde{S} \setminus N^+(x)$ is a solution to $(D', \phi')$ that is smaller than $S'_{opt}$. This is a contradiction and hence the claim is true. □

With this, we are ready to make the final few definitions and conclude the chapter. They involve a function which keeps track of the potential drop corresponding to each vertex being a sink, alongside some sets that are of interest to the algorithm.

**Definition 4.5** (Drop function)**.** *The* drop function*, denoted by* $\psi(x)$ *takes the value* $\psi(x) = \phi(R(x)) + 0.75 * |R^+(x) \setminus R(x)|$ *for every undecided vertex in the given instance.*

**Definition 4.6** (Surviving set)**.** *The* surviving set *of vertex* $x$*, denoted by* $\mathcal{S}_x$ *is defined as the set* $\{u \in N^+(x) \mid N^-(u) \cap \mathcal{U} = \{x\}\}$.

**Definition 4.7** (Candidate-Sink set)**.** *The* candidate-sink set *of vertex* $x$*, denoted by* $\mathcal{C}_x$ *is defined as the set* $\{u \in R^+(y) \mid y \in N^+(x)\}$.

Observe that for a given instance $(D, \phi)$, if $x \in \mathcal{U}$ has a nonempty survivor set, then the vertices of $\mathcal{S}_x$ is in a minimal solution $S_{min}$ if and only if $x$ is in $Z_{min}$ since Corollary 4.2 requires $S_{min} = N^+(Z_{min})$ and $x$ is the only in-neighbour of elements of $\mathcal{S}_x$ which can be in $Z_{min}$.

Also note that, for any given instance $(D, \phi)$ if $x \notin Z_{min}$ then some out-neighbour $y$ of $x$ must satisfy $y \notin S_{min}$. This $y$ must reach some sink $s$ in the final solution, and by definition, $s$ must belong to $R^+(y)$. Thus $\mathcal{C}_x$ is the set of sinks the out neighbours of $x$ can possibly reach if $x$ is not a sink in the final solution. Also, note $\mathcal{C}_x \subseteq N^-(x) \cup R^+(x)$, since given $y \in N^+(x)$ and $s \in R^+(y)$, either $x \in N^+(s)$ or $x$ has a path to $s$ via $y$ in $D - N^+(s)$.

# Chapter 5

# An Improved Exact Algorithm for Knot Free Vertex Deletion

## 5.1 Algorithm KFVD

In this section, we provide an improved exact exponential algorithm (Algorithm 4) to compute the minimum knot-free vertex deletion set. This chapter is based on the paper [21]. We begin by initialising the potential of all vertices to 1 and as soon as we decide a vertex to be non-sink we reduce its potential to 0.25. In our algorithm, whenever we encounter a sink or a source, we remove them using Reduction Rules 2 and 3. If all the vertices have potential 0.25, then we apply Reduction Rule 1 to solve the instance in polynomial time.

At any point, if there exists an undecided vertex $x$ of potential $\psi(x) \geq 3.75$ then we branch on the possibility of it being a sink or non-sink in the optimal solution. Here we benefit from the high potential drop in the branch where $x$ becomes a sink which gives us a branching factor of $(3.75, 0.75)$. Once such vertices are exhausted, we choose an undecided vertex $x$ with the *maximum* number of undecided neighbours to branch on. We show that if $x$ is not a sink in the optimal solution then some other vertex $s$ from $\mathcal{C}_x$ has to be. Further, the bound on $\psi(x)$ helps us limit the cardinality of $\mathcal{C}_x$ and consequently the number of branches. In this case, we get a set of vertices from which atleast one has to be a sink in the final solution. Since, each branch has some vertex becoming a sink, the potential drop will be high enough to give a *good* running time for our algorithm.

**Algorithm 4:** KFVD $(D, \phi)$

**Input:** A directed graph $D$ and a potential function $\phi$
**Output:** The size of a minimum knot-free vertex deletion set

---

**if** $\exists$ *x such that* $N^-(x) = \emptyset$ **then**
  **return** $\mathsf{KFVD}(D - \{x\}, \phi)$;

**if** $\exists$ *x such that* $N^+(x) = \emptyset$ **then**
  **return** $\mathsf{KFVD}(D - R(x), \phi)$;

**if** $V(D) \subseteq \bar{\mathcal{U}}$ **then**
  **return** $|V(D)|$;

**if** $\exists\, x \in \mathcal{U}$ *such that* $\psi(x) \geq 3.75$ **then**
  $D_1, \phi_1 = update(D, \phi, x, -)$;
  $D_2, \phi_2 = update(D, \phi, -, x)$;
  **return** $\min\{\mathsf{KFVD}(D_1, \phi_1) + |N^+(x)|,\ \mathsf{KFVD}(D_2, \phi_2)\}$;

pick $x \in \mathcal{U}$ maximizing $|N(x) \cap \mathcal{U}|$

**if** $\mathcal{S}_x = \emptyset$ **then**
  **for** $s_i \in \mathcal{C}_x$ **do**
    $D_i, \phi_i = update(D, \phi, s_i, -)$
  **return** $\min\{\mathsf{KFVD}(D_x, \phi_x) + |N^+(x)|,\ \min_{s_i}\{\mathsf{KFVD}(D_i, \phi_i) + |N^+(s_i)|\}\}$;
**else**
  **if** $|N(x) \cap \mathcal{U}| \geq 1$ **then**
    $D_x, \phi_x = update(D, \phi, x, -)$;
    pick $y \in \mathcal{S}_x$
    set $B_x = R^+(y)$
    **for** $s_i \in B_x$ **do**
      **if** $N^+(s_i) \subseteq N^+(s_j)$ *for some* $s_j \in B_x$ **then**
        $B_x = B_x - s_j$
    **for** $s_i \in B_x$ **do**
      $NS_i = \{s_j : j < i\}$
      $D_i, \phi_i = update(D, \phi, s_i, NS_i)$
    **return** $\min\{\mathsf{KFVD}(D_x, \phi_x) + |N^+(x)|,\ \min_{s_i}\{\mathsf{KFVD}(D_i, \phi_i) + |N^+(s_i)|\}\}$;
  **else**
    $D_x, \phi_x = update(D, \phi, x, -)$;
    pick $y \in \mathcal{S}_x$, $s \in R^+(y)$
    $D_s, \phi_s = update(D, \phi, s, -)$
    **return** $\min\{\mathsf{KFVD}(D_x, \phi_x) + |N^+(x)|,\ \mathsf{KFVD}(D_s, \phi_s) + |N^+(s)|\}\}$;

The algorithm presented in this chapter, proof of its correctness and upper bound on its running time are all new and make up a major part of the work conducted during the masters thesis [21]. The algorithm draws inspiration from and builds upon *An exact algorithm for knot-free vertex deletion* [18] by MS Ramanujan, Abhishek Sahu, Saket Saurabh and Shaily Verma.

## 5.2 Correctness of the algorithm

Let *Subroutine* 0 represent the lines $1-6$ of the algorithm. The correctness of lines $5-6$, $1-2$ and $3-4$ follow from Reduction Rules 1, 2 and 3 respectively. We use the terms *Subroutine* 1, *Subroutine* 2, *Subroutine* 3 and *Subroutine* 4 to refer to lines 7-10, 12-15, 17-27 and 28-32 of Algorithm KFVD respectively.

### 5.2.1 Correctness of Subroutine 1

**Lemma 5.1.** *If* $\exists\ x \in V(D)$ *such that* $x \in \mathcal{U}$ *and* $\psi(x) \geq 3.75$*, then* $\mathsf{KFVD}(D,\phi) = \min\{\mathsf{KFVD}(D_1,\phi_1) + |N^+(x)|,\ \mathsf{KFVD}(D_2,\phi_2)\}$ *where,* $(D_1,\phi_1) = update(D,\phi,x,-)$ *and* $(D_2,\phi_2) = update(D,\phi,-,x)$.

*Proof.* We prove the lemma using an inductive argument on the potential of the instance ($\phi(D)$). Observe that if we consider the base case $\phi(D) = 1$, then there is only one undecided vertex in the input instance and the recurrence holds true. Now given an instance $D$, assume that the algorithm computes the correct solution for all smaller instances. Let the solutions for $\mathsf{KFVD}(D_1,\phi_1)$ and $\mathsf{KFVD}(D_2,\phi_2)$ be $S_1$ and $S_2$, respectively. Assuming $S_{opt}$ is an optimal solution for $\mathsf{KFVD}(D,\phi)$, we evaluate the two possibilities:

- **Case 1:** $\boldsymbol{x \in Z_{opt}}$. We show that $S_1 \cup N^+(x)$ is an optimal solution for $\mathsf{KFVD}(D,\phi)$ if and only if $S_1$ is an optimal solution for $\mathsf{KFVD}(D_1,\phi_1)$. The arguments are exactly the same as that in Corollary 4.5. We also claim that $\mathsf{KFVD}(D_2,\phi_2) \geq \mathsf{KFVD}(D_1,\phi_1) + |N^+(x)|$. For contradiction, suppose that is not the case, then any optimal solution $S_2$ for $(D_2,\phi_2)$ is also a feasible solution for $\mathsf{KFVD}(D,\phi)$. But $S_2$ has size strictly smaller

than $S_{opt}$, which contradicts our assumption that it is optimal. Hence, $\mathsf{KFVD}(D, \phi) = \min\{\mathsf{KFVD}\ (D_1, \phi) + |N^+(x)|,\ \mathsf{KFVD}(D, \phi_2)\}$.

- **Case 2:** $\boldsymbol{x \notin Z_{opt}}$. In this case, $\mathsf{KFVD}(D_2, \phi_2) = \mathsf{KFVD}(D, \phi)$, by definition. Also $\mathsf{KFVD}(D_2, \phi_2) \leq \mathsf{KFVD}(D_1, \phi_1) + |N^+(x)|$, otherwise we have $S' = S_1 \cup N^+(x)$ as a solution with size strictly smaller than $S_{opt}$ which contradicts our assumption that it is optimal. Hence, $\mathsf{KFVD}(D, \phi) = \min\{\mathsf{KFVD}(D_1, \phi) + |N^+(x)|,\ \mathsf{KFVD}(D, \phi_2)\}$. □

In Subroutine 1, we get a branching vector $(3.75, 0.75)$ since we branch on vertices with $\psi \geq 3.75$. After exhaustively running Subroutine 1, every remaining undecided vertex satisfies $\psi(x) \leq 3.75$. An important implication of this bound is the restricted number of undecided vertices in $N(x)$ as well as $\mathcal{C}_x$, which we prove in the following lemma.

**Lemma 5.2.** *If $(D, \phi)$ is an instance on which Subroutine 1 is no longer applicable, then every $x \in \mathcal{U}$ satisfies, $|\mathcal{C}_x| \leq 2$.*

*Proof.* If Subroutine 1 is no longer applicable to $(D, \phi)$, then every undecided vertex has at most 2 undecided neighbours, since $N(x) \subseteq R(x)$ and $\phi(R(x))$ is counted towards $\psi(x)$.
To begin with, consider the possibility that, $|R^+(x) \setminus R(x)| \geq 3$. Observe that while computing $\psi(x)$, vertices of $|R^+(x) \setminus R(x)|$ contribute a total of 2.25, the potential of $x$ contributes 1 and the in-neighbour and out-neighbour of $x$ has to contribute atleast 0.5. This adds up to a total of 3.75 which is not allowed since Subroutine 1 is no longer applicable. Hence $|R^+(x) \setminus R(x)| \leq 2$. Also, recall that $\mathcal{C}_x \subseteq N^-(x) \cup R^+(x) \subseteq R(x) \cup R^+(x)$. Now we can consider the following possibilities.

- **Case 1:** $\boldsymbol{|R^+(x) \setminus R(x)| = 0}$. Here $\mathcal{C}_x \subseteq R(x)$. Also $\phi(R(x))$ is less than 3.75, out of which $x$ contributes 1. Hence $R(x)$ can have at most 2 more undecided vertices and consequently, $|\mathcal{C}_x| \leq 2$.

- **Case 2:** $\boldsymbol{|R^+(x) \setminus R(x)| = 1}$. In this case, since $R^+(x) \setminus R(x)$ contributes 0.75, the contribution of $R(x)$ to $\psi(x)$ has to be less than 3. Since $x$ itself contributes 1, we can have at most one other undecided vertex in $R(x)$. Hence $|\mathcal{C}_x| \leq 2$.

- **Case 3: $|R^+(x) \setminus R(x)| = 2$**. Here, vertices in $R^+(x) \setminus R(x)$ contribute a total of 1.5 to $\psi(x)$ which gives $\phi(R(x)) < 2.25$. Hence if $|\mathcal{U} \cap R(x)| \geq 2$ then we get $|R(x)| = 2$ which is not possible since $d^+(x), d^-(x) \geq 1$. Hence $\mathcal{U} \cap R(x) = \{x\}$ and $|\mathcal{C}_x| \leq 2$.

□

### 5.2.2 Correctness of Subroutine 2

In Subroutine 2, we deal with undecided vertices that have an empty surviving set corresponding to them. The fact that every out-neighbour of such a vertex has some other undecided in-neighbour helps us establish a high potential for elements of $\mathcal{C}_x$.

**Lemma 5.3.** *Let $(D, \phi)$ be an instance such that $\psi(v) < 3.75$, for every vertex in $\mathcal{U}$. let $x$ be a vertex maximizing $\mathcal{U} \cap N(x)$ and $\mathcal{S}_x = \emptyset$. We claim,*

$$\mathsf{KFVD}(D, \phi) = min\{\mathsf{KFVD}(D_x, \phi_x) + |N^+(x)|,\ min_{s_i}\{\mathsf{KFVD}(D_i, \phi_i) + |N^+(s_i)|\}\}$$

*where $(D_x, \phi_x) = update(D, \phi, x, -)$ and $(D_i, \phi_i) = update(D, \phi, s_i, -)$ for every $s_i \in \mathcal{C}_x$.*

*Proof.* In any given minimal solution, either $x$ is a sink or at least one of its out-neighbours must survive. The out-neighbour which survives, say $y$, must be able to reach a sink in the final solution. This sink by definition, belongs to $R^+(y)$. Hence, if $x$ is not a sink in the final solution, then at least one vertex of $\mathcal{C}_x$ has to be. Thus, every possible minimal solution contains at least one element from the set $\{x\} \cup \mathcal{C}_x$ in its sink set. By induction, assuming that $\mathsf{KFVD}(D', \phi')$ returns the optimal solution for every instance smaller than $(D, \phi)$ along with Corollary 4.5, proves the lemma. □

Observe that, since $|\mathcal{C}_x| \leq 2$, we have at most 3 branches when running Subroutine 2. Further we claim that whenever we branch on the case where $s_i \in \mathcal{C}_x$ becomes a sink, the potential drop is at least 3. We have, either $x \in N^+(s_i)$ or $x \in R^-(s_i)$. Further, $s_i \in R^+(y)$ for some $y \in N^+(x)$ and since $\mathcal{S}_x = \emptyset$, $y$ has some undecided in-neighbour $z$ different from $x$. Note that, $z \neq s_i$ since $s_i \in R^+(y)$. Similar to $x$, $z \in N^+(s_i)$ or $z \in R^-(s_i)$. Hence $\psi(s_i) \geq \phi(\{s_i, x, z\}) = 3$. Now let us look at the possible branches which could arise and their corresponding worst case branching factor. Note that if $|\mathcal{C}_x| = 0$, no branching is involved and the subroutine is executed as a reduction rule.

**Note:** In the following sections we use diagrams to give some intuitions about the cases we are considering. In them, white vertices are undecided, yellow ones are semi-decided. Dotted lines indicate possibility of in and out-neighbours, dashed lines denote directed paths and thick lines denote edges.

- **Case 1: $\mathcal{C}_x = \{s\}$**. Here, we have the following possibilities:
  - $|N(x) \cap \mathcal{C}_x| = 0$, in which case, $N[x]$ has potential at least 1.5, since $x$ has atleast two neighbours and $\phi(x) = 1$. Further, $s \in \mathcal{C}_x$ contributes 0.75, giving $\psi(x) \geq 2.25$.

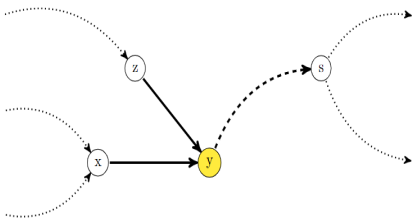


Figure 5.1: Case 1.1: $N[x] \cap \mathcal{C}_x = \emptyset$

  - $|N(x) \cap \mathcal{C}_x| = 1$, in which case, $N[x]$ has potential at least 2.25, since $\phi(x) = \phi(s) = 1$, and $x$ has at least one more neighbour, giving $\psi(x) \geq 2.25$.

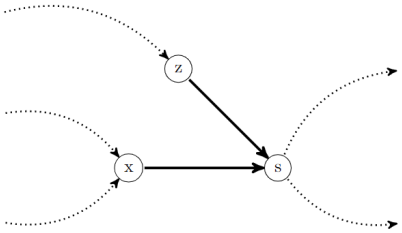


Figure 5.2: Case 1.2: $s \in N[x] \cap \mathcal{C}_x$

- **Case 2: $\mathcal{C}_x = \{s_1, s_2\}$**. Here the possibilities are as follows.
  - $|N(x) \cap \mathcal{C}_x| = 0$, in which case, $N[x]$ has potential at least 1.5, since $x$ has at least two neighbours and $\phi(x) = 1$. Further $s_1, s_2 \in \mathcal{C}_x$ contribute 1.5, giving $\psi(x) \geq 3$.

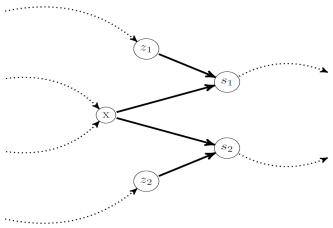


Figure 5.3: Case 2.1: $s_1, s_2 \in N[x] \cap \mathcal{C}_x$

– $N(x) \cap \mathcal{C}_x = \{s_1\}$, in which case, $N[x]$ has potential at least 2.25, since $x$ has at least two neighbours, including $s_1$. Further $s_2 \in \mathcal{C}_x \setminus N(x)$ contributes 0.75, giving $\psi(x) \geq 3$.

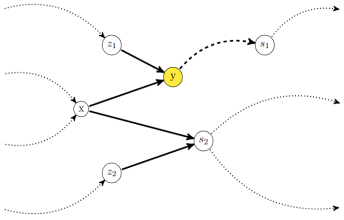


Figure 5.4: Case 2.2: $s_1 \in N[x] \cap \mathcal{C}_x$

– $|N(x) \cap \mathcal{C}_x| = 2$, in which case, $N[x]$ has potential at least 3, due to $\phi(x) = \phi(s_1) = \phi(s_2) = 1$ and hence $\psi(x) \geq 3$.

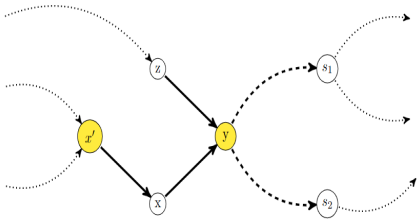


Figure 5.5: Case 2.3: $N[x] \cap \mathcal{C}_x = \emptyset$

Hence the worst case branching vectors from Subroutine 2 are $(3, 2.25)$ and $(3, 3, 3)$.

### 5.2.3 Correctness of Subroutine 3

In Subroutine 3 we branch on undecided vertices with a non-empty surviving set and at least one undecided neighbour. Let us define $D_x = \{u \mid \exists\ v \in R^+(y),\ v \neq u,\ N^+(v) \subseteq N^+(u)\}$ and $B_x$ to be $R^+(y) \setminus D_x$. We also fix an arbitrary ordering of vertices of $B_x$ and define $NS_i = \{s_j \in B_x \mid j < i\}$

**Lemma 5.4.** *Let $(D, \phi)$ be an instance such that $\psi(v) < 3.75$, for every vertex in $\mathcal{U}$, let $x$ be a vertex maximizing $\mathcal{U} \cap N(x)$ and $y \in \mathcal{S}_x$. We claim*

$$\mathsf{KFVD}(D, \phi) = min\{\mathsf{KFVD}(D_x, \phi_x) + |N^+(x)|,\ min_{s_i}\{\mathsf{KFVD}(D_i, \phi_i) + |N^+(s_i)|\}\}$$

*where $(D_x, \phi_x) = update(D, \phi, x, -)$ and $(D_i, \phi_i) = update(D, \phi, s_i, NS_i)$ for all $s_i \in B_x$.*

*Proof.* We claim that $y \in \mathcal{S}_x$ survives in every minimal solution where $x$ is not a sink. We have $y \in S_{min}$ if and only if some in-neighbour of $y$ is a sink in $D - S_{min}$, but $y \in \mathcal{S}_x$ implies $N^-(y) \cap \mathcal{U} = \{x\}$. Hence, if $x \notin Z_{min}$, then $y \notin S_{min}$ and we can assume that if $x$ is not a sink then $y$ survives and must reach some sink. This sink by definition must be in $R^+(y)$. But if some distinct $u, v$ in $R^+(y)$ satisfy $N^+(v) \subseteq N^+(u)$, then in every minimal solution where $u$ is a sink, $v$ is also a sink and $\mathsf{KFVD}(update(D, \phi, v, -)) \leq \mathsf{KFVD}(update(D, \phi, u, -))$. Thus, every possible minimal solution contains at least one element from the set $\{x\} \cup (R^+(y) \setminus D_x) = \{x\} \cup B_x$ in its sink set. Further, we can fix an arbitrary ordering for elements of $B_x$ and branch on the first vertex of this ordering which becomes a sink. In the branch where the $i^{th}$ vertex becomes a sink, we can assume that all the vertices before it are non-sinks. Hence by induction, assuming that $\mathsf{KFVD}(D', \phi')$ returns the optimal solution for every instance smaller than $(D, \phi)$, proves the claim. □

Since $\psi(x) < 3.75$, $N(x)$ has at most two undecided vertices whenever $x$ is undecided. We will analyze the branching vector for this subroutine in two steps. For undecided vertices with two undecided neighbours and for undecided vertices with one undecided neighbour. If $R^+(y)$ is empty, then $x$ must be a sink and no branching is involved. Also, Lemma 5.2 implies $R^+(y)$ can have cardinality at most 2, which leads to the following possibilities.

## $B_x = \{s\}$

Observe that if $x$ is a sink, then since $x$ has at least three vertices in its closed neighbourhood, at least two of which are undecided, we have $\psi(x) \geq 2.25$. But, if $x$ is not a sink and $s$ is the sink $y$ reaches, then since $s$ has at least two neighbours, and $x \in N^+(s)$ or $x \in R^-(s)$, we have $\psi(s) \geq \phi(N[s] \cup \{x\}) \geq 2.25$. Hence the worst case branching vector is $(2.25, 2.25)$.

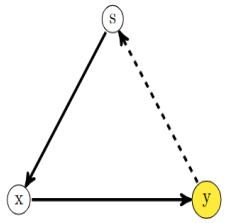


Figure 5.6: Case 1: $B_x = \{s\}$

## $B_x = \{s_1, s_2\}$

In this case, depending on the number of undecided neighbours $x$ has, we have the following cases.

**Case 1: Branching on a vertex with two undecided neighbours**

Here $|N(x) \cap \mathcal{U}| = 2$ implies $\phi(N[x]) \geq 3$ and hence $B_x \setminus N(x) = \emptyset$. Otherwise, the vertex in $B_x \setminus N(x)$ would contribute 0.75 to $\psi(x)$ making it at least 3.75 which contradicts our assumption that Subroutine 1 is no longer applicable. Now, assuming $y \in \mathcal{S}_x$ is the chosen vertex, we have the following possibilities.

- **Subcase 1.1: $s_2 \in R^-(s_1)$ or $s_1 \in R^-(s_2)$**

  The arguments for both cases are identical, hence assume that $s_2 \in R^-(s_1)$.

  - If $x$ is a sink, then since $x$ has at least three undecided vertices in its closed neighbourhood each contributing 1 giving $\psi(x) \geq 3$.
  - If $x$ is not a sink and $s_1$ is a sink which reaches in the final solution. Then we have $x \in N(s_1)$, $s_2 \in R^-(s_1)$ and hence $\psi(s_1) \geq 3$

- If $x$ is not a sink, $s_1$ is not a sink in the final solution but $s_2$ is. Then we have $x \in N(s_2)$, $y \in R^-(s_2)$ and $s_1 \in R^+(s_2)$. Now if $s_1 \in R^-(s_2)$ then $x, s_1$ and $s_2$ each contribute 1 to $\psi(s_2)$. Else, we can assume $y \neq s_1$ and $x, y, s_2$ contributes a total of 2.25 and $s_1$ contributes 0.75 giving $\psi(s_2) \geq 3$.

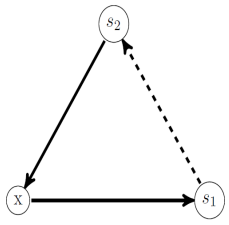


Figure 5.7: Case 2.1.1: $s_2 \in R^-(s_1)$

Hence we have a worst case branching vector $(3, 3, 3)$. Further, in the rest of the sub-cases, we can assume $s_1 \notin R^-(s_2)$ and vice-versa which implies $y \notin \{s_1, s_2\}$. Now consider the case where $s_1 \in N^+(x),\ s_2 \in N^-(x)$, since $x \notin N^+(s_1)$, either $s_2 \in N^+(s_1)$ or $s_2 \in R^-(s_1)$. But $s_2 \notin R^-(s_1)$ is and $s_2 \in N^+(s_1)$ implies $s_1 \in R^-(s_2)$, which cannot be the case. Thus this possibility is already considered, which leaves us with the following configurations.

- **Subcase 1.2: $s_1, s_2 \in N^-(x)$**

  - If $x$ is a sink, since $N[x]$ has at least four vertices out of which three are undecided, we have $\psi(x) \geq 3.25$.
  - If $x$ is not a sink and $s_1$ is a sink which $y$ reaches in the final solution. Then we have $x \in N^+(s_1)$, $y \in R^-(s_1)$. Also, since $N^+(s_1) \not\subseteq N^+(s_2)$ and $x \in N^+(s_2)$, $N^+(s_1)$ has at least one more vertex other than $x$, giving $\psi(s_1) \geq 2.5$.
  - If $x$ is not a sink, $s_1$ is not a sink in the final solution, but $s_2$ is. Then we have $x \in N^+(s_2)$, $y \in R^-(s_2)$. Also, since $N^+(s_2) \not\subseteq N^+(s_1)$ and $x \in N^+(s_1)$, $N^+(s_2)$ has at least one more vertex, giving $\phi(R(s_2)) \geq 2.5$. Further, since we also have the added information that $s_1$ is not a sink which gives a drop of 0.75, the total drop in potential is at least 3.25.

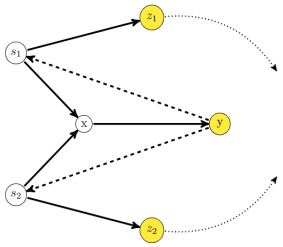


Figure 5.8: Case 2.1.2: $s_1, s_2 \in N^-(x)$

Here we have a worst case branching vector $(2.5, 3.25, 3.25)$

- **Subcase 1.3: $s_1, s_2 \in N^+(x)$**

  - If $x$ is a sink, since $N[x]$ has at least four vertices out of which three are undecided, we have $\psi(x) \geq 3.25$.
  - If $x$ is not a sink and $s_1$ is a sink which $y$ reaches in the final solution. Then we have $x \in N^-(s_1)$, $y \in R^-(s_1)$ and at least one other vertex in $N^+(s_1)$ giving $\psi(s_1) \geq 2.5$.
  - If $x$ is not a sink, $s_1$ is not a sink in the final solution, but $s_2$ is. Then we have, $x \in N^-(s_2)$, $y \in R^-(s_2)$ and at least one other vertex in $N^+(s_2)$ giving $\psi(s_2) \geq 2.5$. Further, we also have the added information that $s_1$ is not a sink which gives a drop of 0.75. Hence, the total drop in potential is at least 3.25.

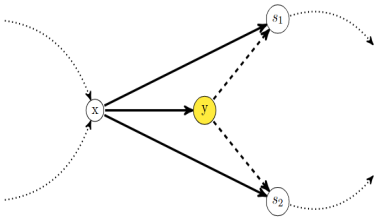


Figure 5.9: Case 2.1.3: $s_1, s_2 \in N^+(x)$

Here we have a worst case branching vector $(3.25, 2.5, 3.25)$.

**Case 2: Branching on a vertex with one undecided neighbour**

Here, $|N(x) \cap \mathcal{U}| = 1$ implies $\phi(N[x]) \geq 2.25$ and hence, $|B_x \setminus N(x)| \leq 1$. Otherwise, the

vertices in $B_x \setminus N(x)$ would contribute 0.75 to $\psi(x)$ making it at least 3.75 which contradicts our assumption that Subroutine 1 is no longer applicable. Now, assuming $s_1 \in N(x)$ and $y \in \mathcal{S}_x$ is the chosen vertex, we have the following possibilities.

- **Subcase 2.1: $s_1 \in R^-(s_2)$** or **$s_2 \in R^-(s_1)$**

  The arguments for both the case are identical, hence we assume $s_1 \in R^-(s_2)$.

  - If $x$ is a sink, then since $x$ has at least three vertices in its closed neighbourhood exactly two of which are undecided and $s_2 \in R^+(x)$, we get $\psi(x) \geq 3$.
  - If $x$ is not a sink and $s_1$ is a sink which $y$ reaches in the final solution. Then we have at least three vertices in $N[s_1]$ out of which only $x,\ s_1$ are undecided. This and $s_2 \in R^+(s_1)$ give $\psi(s_1) \geq 3$.
  - If $x$ is not a sink, $s_1$ is not a sink in the final solution but $s_2$ is. Then we have $x \in R^-(s_2)$ and $s_1 \in R^-(s_2)$. Now, $s_2$ has an out-neighbour which cannot be $x$ since $x$ has only one undecided neighbour and $s_1$ since $s_2 \in R^+(s_1)$. This out-neighbour contributes at least 0.25 making the total drop at least 3.25.

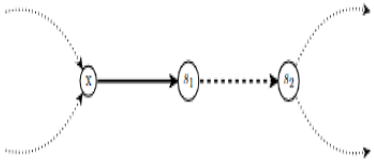


Figure 5.10: Case 2.2.1: $s_1 \in R^-(s_2)$

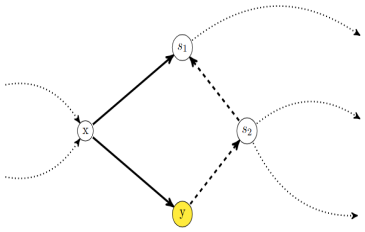


Figure 5.11: Case 2.2.1: $s_2 \in R^-(s_1)$

Hence the worst case branching vector is $(3, 3, 3.25)$.

- **Subcase 2.2: $s_2 \notin R^-(s_1)$ & $s_1 \notin R^-(s_2)$**

  Here, $s_1 \notin N^-(x)$, since in that case $s_1 \in R^-(s_2)$ or $s_1 \in N^+(s_2)$ which implies $s_2 \in R^-(s_1)$. Also, $y \neq s_1$ since otherwise $s_1 \in R^-(s_2)$.

  - If $x$ is a sink, then since $x$ has at least two out-neighbours $y, s_1$ and some in-neighbour $z$, along with $s_2 \in R^+(x)$ we have $\psi(x) \geq 3.25$.

- If $x$ is not a sink and $s_1$ is a sink which $y$ reaches. Then we have $x, y$ in $R^-(s_1)$ and since $s_1$ has some other out-neighbour, we get $\psi(s_1) \geq 2.5$.
- If $x$ is not a sink, $s_1$ is not a sink in the final solution but $s_2$ is. Then we have, $x, y, z$ in $R^-(s_2)$ which gives $\psi(s_2) \geq 2.5$. Further, since we also have the added information that $s_1$ is not a sink which gives a drop of 0.75, the total drop in potential is at least 3.25.

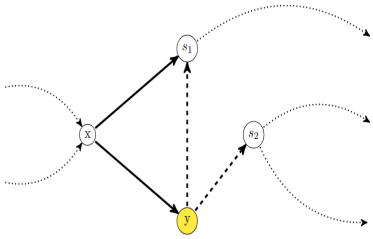


Figure 5.12: Case 2.2.2: $s_1, s_2 \in N^+(x)$

Hence the worst case branching vector is $(3.25, 2.5, 3.25)$.

### 5.2.4 Correctness of Subroutine 4

In Subroutine 4, we branch on undecided vertices with a non-empty surviving set and zero undecided neighbour. It is executed only when Subroutines $1, 2$ and 3 is no longer applicable. Hence, we can assume all the vertices no longer satisfy the requirements of Subroutine $1, 2$ and 3. In particular, note that every undecided vertex has zero undecided neighbours.

**Lemma 5.5.** *Let $(D, \phi)$ be an instance such that $\psi(v) < 3.75$ and $\mathcal{U} \cap N(v) = \emptyset$ for every vertex in $\mathcal{U}$. Let $x$ be an undecided vertex, $y \in \mathcal{S}_x$ and $s \in R^+(y)$. We claim*

$$\mathsf{KFVD}(D, \phi) = min\{\mathsf{KFVD}(D_x, \phi_x) + |N^+(x)|,\ \mathsf{KFVD}(D_s, \phi_s) + |N^+(s)|\}$$

*where $(D_x, \phi_x) = update(D, \phi, x, -)$ and $(D_s, \phi_s) = update(D, \phi, s, -)$.*

*Proof.* We claim that $y \in \mathcal{S}_x$ survives in every minimal solution where $x$ is not a sink. We have, $y \in S_{min}$ if and only if some in-neighbour of $y$ is a sink in $D - S_{min}$, but $y \in \mathcal{S}_x$ implies $N^-(y) \cap \mathcal{U} = \{x\}$. Hence, if $x \notin Z_{min}$, then $y \notin S_{min}$. Now, assuming $x$ is not a sink, $y$ survives, and it must reach some sink, which by definition must belong to $R^+(y)$. Now

assume that $s \in R^+(y)$ is not a sink in the final solution where $x$ is not a sink and $y$ survives. Since $s$ is not a sink, it must either belong to the solution set or reach a sink vertex. But $s$ cannot belong to any minimal feasible solution to $(D, \phi)$, since $N^-(s) \cap \mathcal{U} = \emptyset$. Hence we can assume $s$ reaches some sink $z$. Now there are two possibilities either $N^+(z)$ intersects a vertex of $P_{(y,s)}$ or not.

- **Case 1: $w_1 \in N^+(z) \cap P_{(y,s)}$.** Here, $z$ has a path to $s$ in $D - N^+(s)$ via $P_{(w_1,s)}$. Also, $x, z \notin N(s)$ since $s$ is undecided and cannot have undecided neighbours and hence $x \in R^-(s)$ since it has a path to $s$ via $y$. Now, $w_1 \neq y$, since $z$ is a sink that $s$ can reach in a solution where $y$ survives and hence $y, w_1 \in R^-(s)$. Thus, $R^-(s)$ contains $x, y, w_1, z$ and $s$. Now, out-neighbour of $s$ in $P_{(s,z)}$ cannot be $y$ or $w_1$ since $N^+(s)$ does not intersect $P_{(y,s)}$ and $y, w_1 \in P_{(y,s)}$. Thus $N^+(s)$ contains at least one semidecided vertex $w_2$ different from $y, w_1$ which implies that $x, y, w_1, z, w_2$ and $s$ belongs to $R(s)$. This leads to a contradiction since this implies $\phi(R(s)) \geq 3.75$.

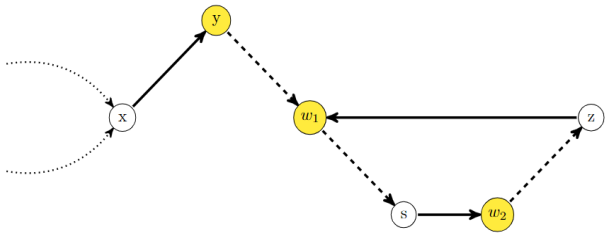


Figure 5.13: Case 1: $w_1 \in N^+(z) \cap P_{(y,s)}$

- **Case 2: $N^+(z) \cap P_{(y,s)} = \emptyset$.** Here, $y$ can reach $z$ in $D - N^+(z)$ via the path $P_{(y,s)}$ followed by $P_{(s,z)}$. Observe that $x \notin N^+(z)$ as both are undecided, and $y \in R^-(z)$ which implies that $x \in R^-(z)$. Now, out-neighbour of $s$ in $P_{(s,z)}$ cannot be $y$ since $N^+(s)$ does not intersect $P_{(y,s)}$ and $y \in P_{(y,s)}$. Thus $P_{(s,z)}$ contains at least one semidecided vertex $w_1$ different from $y$ and $R^-(z)$ contains $x, y, w_1, z$ and $s$. Now, out-neighbour of $z$ cannot be $y$, since $z$ is a sink that $s$ can reach in a solution where $y$ survives and it cannot be $w_1$ since $N^+(z)$ does not intersect $P_{(s,z)}$ and $w_1 \in P_{(s,z)}$. Thus $N^+(z)$ contains at least one semidecided vertex $w_2$ different from $y, w_1$ which implies that $x, y, w_1, z, w_2$ and $s$ belong to $R(z)$. This leads to a contradiction since, $\phi(R(z)) \geq 3.75$.

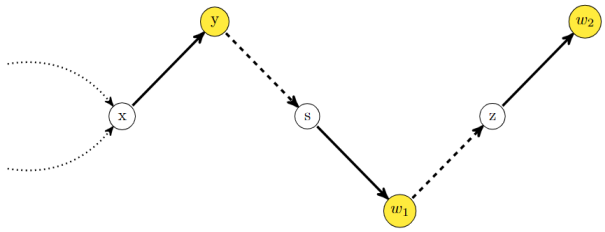


Figure 5.14: Case 2: $N^+(z) \cap P_{(y,s)} = \emptyset$

Observe that both possibilities lead to a contradiction and hence our assumption that $s$ reaches some sink $z$ in a minimal solution where $x$ is not a sink and $y$ survives, has to be wrong. Thus, for the given instance where Subroutines 1, 2 and 3 are no longer applicable, in any minimal solution where $x$ is not a sink, $y \in \mathcal{S}_x$ and $s \in R^+(y)$, $s$ has to be a sink. By induction, assuming that $\mathsf{KFVD}(D', \phi')$ returns the optimal solution for every instance smaller than $(D, \phi)$, proves the claim. □

If $R^+(y) = \emptyset$ then $x$ has to be a sink and the subroutine is executed as a reduction rule without branching. Now, since $x$ has at least two neighbours and as $s \in R^+(x)$ we get $\psi(x) \geq 2.25$. Similarly, since $s$ has at least two neighbours and $x \in R^-(s)$ we get $\psi(s) \geq 2.5$. Hence the worst case branching factor is $(2.25, 2.5)$.

## 5.3 Running time analysis

Observe that the reduction Rules 1, 2 and 3 can be applied on any input instance in polynomial time. The branching vector obtained in Subroutine 1 was $(3.75, 0.75)$, which gives the recurrence $f(\mu) \leq f(\mu - 3.75) + f(\mu - 0.75)$. This solves to $f(\mu) = \mathcal{O}(1.4549^\mu)$. For Subroutine 2 we observe a worst case drop in measure of $(2.25, 3)$ and $(3, 3, 3)$, amongst which $(3, 3, 3)$ has the higher running time $f(\mu) = \mathcal{O}(1.4422^\mu)$. Similarly, for Subroutine 3, we have branching vectors $(2.25, 2.25)$, $(3.25, 2.5, 3.25)$, and $(3, 3, 3)$, out of which $(3.25, 2.5, 3.25)$ gives the worst running time $f(\mu) = \mathcal{O}(1.4465^\mu)$. Finally Subroutine 4 has a branching vector $(2.25, 2.5)$ which gives $f(\mu) = \mathcal{O}(1.3393^\mu)$.

For an input instance $D$ of Knot-Free Vertex Deletion, we initialise the potential of

each vertex to 1 and run KFVD$(D, \phi)$. Observe that, $\phi(V(D)) = |V(D)| = n$ and during the recursive calls the potential of the instance never increases or drops below 0. Hence we can bound the running time of the algorithm by the run time corresponding to its worst case subroutine which is Subroutine 1. This gives us the following theorem.

**Theorem 5.6.** *Algorithm* KFVD *solves* Knot-Free Vertex Deletion *in* $\mathcal{O}^*(1.4549^n)$.

# Chapter 6

# Complimentary Results

In this chapter, we prove some results to complement the algorithm presented in the previous chapter. We present a lower bound for the running time of our algorithm, alongside an upper bound for the number of inclusion wise minimal knot-free vertex deletion sets in a graph of order $n$. These results are new and make up a part of the work conducted during the masters thesis [21].

## 6.1 Lower bound on the run-time of our algorithm

We run our algorithm KFVD on the graph $D$ in figure 6.1, where $V(D) = \{a_i, b_i, c_i \mid 1 \leq i \leq \frac{n}{3}\}$ and $E(D) = \{(a_i, b_i), (b_i, c_i), (c_i, a_i) \mid 1 \leq i \leq \frac{n}{3}\}$.

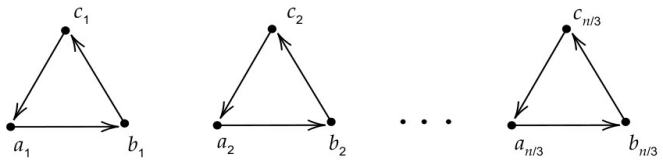


Figure 6.1: Illustration of a worst-case instance for our algorithm.

We claim that in the worst case, our algorithm takes $\mathcal{O}^*(3^{n/3})$ time to solve KFVD on $D$

which we prove via adversarial arguments. Initially since $\phi(v) = 1$ for every vertex of $D$, we have $\psi(a_i) = 3$, $\psi(b_i) = 3$, $\psi(c_i) = 3$. Since $\psi(v) < 3.75$ and $\mathcal{S}_v \neq \emptyset$ for every vertex, Subroutine 1 and 2 are not applicable. The adversary chooses the vertex $a_i$ to branch on. Since $b_i \in \mathcal{S}_{a_i}$ and $b_i, c_i \in R^+(b_i)$ we get one branch where $a_i$ becomes a sink with potential drop 3. Another where $a_i$ is not a sink, but $b_i$ is, with drop 3. Finally, one where $a_i, b_i$ are not sinks, but $c_i$ is, with drop 3.

In all of the above branches, $\{a_i, b_i, c_i\}$ is removed, giving a recurrence relation $T(n) = 3T(n-3)$ which implies $T(n)$ is $3^{n/3} \geq 1.4422^n$ for this instance. Which gives us the following theorem.

**Theorem 6.1.** *Algorithm* KFVD *runs in time* $\Omega(1.4422^n)$.

## 6.2 Upper bound on number of minimal solutions

We claim that if we run our algorithm on any given directed graph, and create a decision tree then for every minimal knot-free vertex deletion set there exists a leaf node of the decision tree which corresponds to it. Note that any algorithm which finds all the minimal solutions needs at least unit time to find each solution and hence the number of minimal solutions for any graph of size $n$ cannot exceed the complexity of the algorithm. This observation gives us the following theorem.

**Theorem 6.2.** *The number of inclusion minimal knot-free vertex deletion sets is* $\mathcal{O}^*(1.4549^n)$.

*Proof.* Let $S_{min}$ be a minimal solution and $Z_{min}$ be the set of sinks in $D - S_{min}$. Beginning with the root of decision tree use the following set of rules to find the corresponding leaf node. If the node corresponds to an execution of subroutine 1 on a vertex $v$, then if $v \in Z_{min}$ choose the branch of the tree where $v$ is added to the sink set, else choose the branch where $v$ is labelled as a non-sink vertex. If the node corresponds to any other subroutine, then by correctness of the algorithm proven in the earlier section, at that node it branches on a set of vertices, which intersects the sink set of any minimal solution. Choose a branch corresponding to a vertex $s$ such that $s \in Z_{min}$. Follow this procedure to get to a leaf node which corresponds to a knot-free vertex deletion set $S_{leaf}$ and sink set $Z_{leaf}$. Note that due to our choice of leaf node, we have $Z_{leaf} \subseteq Z_{min}$ and consequently $S_{leaf} \subseteq S_{min}$. This along

with minimality of $S_{min}$ gives $S_{leaf} = S_{min}$. Hence the number of minimal knot-free vertex deletion sets of $D$ cannot exceed the number of leaf nodes of the decision tree corresponding to any run of the algorithm on $D$. Hence maximum number of minimal knot-free vertex deletion sets is $\mathcal{O}^*(1.4549^n)$. □

**Theorem 6.3.** *There exists an infinite family of graphs with* $\Omega(1.4422^n)$ *many inclusion-wise minimal knot-free vertex deletion sets.*

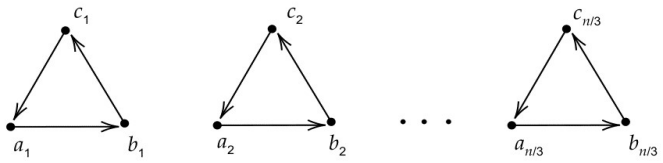


Figure 6.2: Family of graphs with $\Omega(1.4422^n)$ minimal knot-free vertex deletion sets

*Proof.* Consider the graph in Figure 6.2. Each strongly connected component $\{a_i, b_i, c_i\}$ can be made knot-free by deleting a single vertex. Hence every set $S$ which contains only one element from $\{a_i, b_i, c_i\}$ is a knot-free vertex deletion set. Observe that there are $3^{\frac{n}{3}}$ many of them since we can choose the element for each $i$ in 3 ways and $i$ ranges from 1 to $\frac{n}{3}$. Further, any proper subset $S'$ of such a set will not intersect $\{a_i, b_i, c_i\}$ for some $i$, leaving $D - S'$ with at least one knot. Hence the graph in Figure 6.2 has at least $3^{\frac{n}{3}} \geq 1.4422^n$ many minimal knot-free vertex deletion sets. Now, by taking graphs which are disjoint union of triangles, we obtain an infinite family of graphs such that each element of that family has at least $1.4422^n$ many minimal knot-free vertex deletion sets. □

# Chapter 7

# Conclusion

In this thesis, we obtain a $\mathcal{O}(1.4549^n)$ time algorithm for the KFVD problem which uses polynomial space. We also obtained a family of graphs for which our algorithm takes $\Omega(1.4422^n)$ time. We further proved an upper bound of $\mathcal{O}(1.4549^n)$ on the number of minimal knot-free vertex deletion sets possible for any directed graph and observed that this bound is nearly optimal by presenting a family of graphs which have $1.4422^n$ many minimal knot-free vertex deletion sets.

Our algorithm is not proven to be optimal and it would be interesting to see what ideas can help improve it. Closing the gap between the upper and lower bound for the maximum number of knot-free vertex deletion sets is also a problem of interest for the future.

# Bibliography


[1] Alan Diêgo Aurélio Carneiro, Fábio Protti, and Uéverton Souza. On knot-free vertex deletion: Fine-grained parameterized complexity analysis of a deadlock resolution graph problem. *Theoretical Computer Science*, 909, 01 2022.

[2] Stéphane Bessy, Marin Bougeret, Alan Diêgo A. Carneiro, Fábio Protti, and Uéverton S. Souza. Width parameterizations for knot-free vertex deletion on digraphs. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, volume 148 of *LIPIcs*, pages 2:1–2:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[3] Andreas Björklund. Determinant Sums for Hamiltonicity (Invited Talk). In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:1, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[4] Alan Diêgo Carneiro, Fábio Protti, and Uéverton S. Souza. Deadlock resolution in wait-for graphs by vertex/arc deletion. *J. Comb. Optim.*, 37(2):546–562, feb 2019.

[5] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 177–186, New York, NY, USA, 2008. Association for Computing Machinery.

[6] Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, page 764–775, New York, NY, USA, 2016. Association for Computing Machinery.

[7] Fedor V. Fomin, Fabrizio Grandoni, Artem V. Pyatkin, and Alexey A. Stepanov. Bounding the number of minimal dominating sets: A measure and conquer approach. In *Proceedings of the 16th International Conference on Algorithms and Computation*, ISAAC'05, page 573–582, Berlin, Heidelberg, 2005. Springer-Verlag.

[8] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[9] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM National Meeting*, ACM '61, page 71.201–71.204, New York, NY, USA, 1961. Association for Computing Machinery.

[10] Gordon Hoi. An improved exact algorithm for the exact satisfiability problem. In Weili Wu and Zhongnan Zhang, editors, *Combinatorial Optimization and Applications*, pages 304–319, Cham, 2020. Springer International Publishing.

[11] Richard Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, volume 40, pages 85–103, 01 1972.

[12] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison Wesley, 2006.

[13] Carlos VGC Lima, Fábio Protti, Dieter Rautenbach, Uéverton S Souza, and Jayme L Szwarcfiter. And/or-convexity: a graph convexity based on processes and deadlock models. *Annals of Operations Research*, 264:267–286, 2018.

[14] Daniel Lokshtanov, Pranabendu Misra, Joydeep Mukherjee, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. 2-approximating feedback vertex set in tournaments. *ACM Trans. Algorithms*, 17(2), apr 2021.

[15] Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. *FPT-approximation for FPT Problems*, pages 199–218. Association for Computing Machinery, 2021.

[16] J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, 1965.

[17] Fabiano de S. Oliveira and Valmir C. Barbosa. Revisiting deadlock prevention: A probabilistic approach. *Networks*, 63(2):203–210, 2014.

[18] MS Ramanujan, Abhishek Sahu, Saket Saurabh, and Shaily Verma. An exact algorithm for knot-free vertex deletion. In *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, 2022.

[19] Igor Razgon. Computing minimum directed feedback vertex set in $\mathcal{O}(1.9977^n)$. In *Italian Conference on Theoretical Computer Science*, 2007.

[20] Kenneth Rosen. *Discrete Mathematics and Its Applications*. Mcgraw-hill, 01 2007.

[21] Ajaykrishnan E. S., Soumen Maity, Abhishek Sahu, and Saket Saurabh. An improved exact algorithm for knot-free vertex deletion. *CoRR*, abs/2303.10866, 2023.

[22] G.J. Woeginger. Exact algorithms for np-hard problems : A survey. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization - Eureka! You shrink! Papers dedicated to Jack Edmonds. (5th International Workshop, Aussois, France, March 2001, Revised papers)*, Lecture Notes in Computer Science, pages 185–207, Germany, 2003. Springer.